Principles of Program Analysis

Lecture 1

Harry Xu

Spring 2013

An Imperfect World

- Software has bugs
  - The northeast blackout of 2003, affected 10 million people in Ontario and 45 million in eight U.S. states (caused by a race condition)
  - The explosion of the Ariane 5, valued at $500 million, 45 seconds after its lift-off (due to an 16-bit integer overflow)
- Software is slow
  - the conversion of a single date field from a SOAP data source to a Java object can require as many as 268 method calls and the generation of 70 objects

Program Analysis

- Discovering facts about programs
- A wide variety of applications
  - Finding bugs (e.g., model checking, testing, etc.)
  - Optimizing performance (e.g., compiler optimizations, bloat detection, etc.)
  - Detecting security vulnerabilities (e.g., detecting violations of security policies, etc.)
  - Improving software maintainability and understandability (e.g., reverse-engineering of UML diagrams, software visualization, etc.)

Static v.s. Dynamic Analysis

- Static analysis
  - Attempt to understand certain program properties without running a program
  - Make over-conservative claims
- Dynamic analysis
  - Need to run user *instrumented* code
  - Add overhead to running time and memory consumption

This Class

- Focus on *static program analysis* in this class
- We will discuss
  - Both principles and practices
  - Both classical program analysis algorithms and the state-of-the-art research
- We will cover five major topics
  - Dataflow analysis
  - Abstract interpretation
  - Constraint-based analysis
  - Type and effect system
  - Scalable interprocedural analysis

This Class

- We will spend two weeks on each topic
  - Discuss analysis principles in the first week (via lectures)
  - Discuss state-or-the-art research in the second week (via student presentations)
- Homework for each topic
  - A project that implements program analysis algorithms in Java
  - Paper critiques
- Students volunteer to present papers
  - 15 slots
  - Bonus credits!

Projects

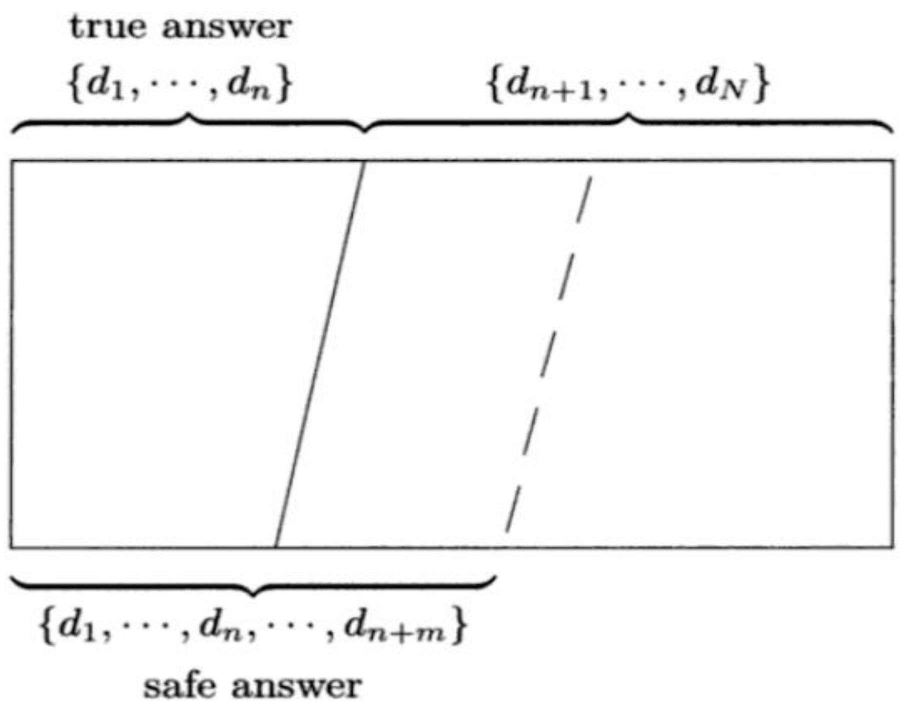
- Two students form a group
- Based on the soot program analysis framework (http://www.sable.mcgill.ca/soot/)
- The first project
  - Implement a “hello-world” version of an intra-procedural analysis that prints out all heap load/store operations
  - Due Friday April 10

Course Pre-Reqs and Grading

- Office hour: Thursday 2—4pm, DBH 3212
- Reader: Taesu Kim
- Prerequisites: Java programming experience
- Grading
  - Paper critiques (20%)
  - Projects (40%)
  - In-class final (40%)

Static Analysis

- Key property: safe approximation
  - A larger set of possibilities than what will ever happen during any execution of the program



A Simple Example

```
read(x);
if(x>0) y = 1;
else {y = 2;  S};   //S does not write y
z = y;
```

A Simple Example

```
read(x);
if(x>0) y = 1;
else {y = 2;  S};   //S does not write z
z = y;
```

- Which of the following statements about *z* are valid from the perspective of a static analysis?

A Simple Example

```
read(x);
if(x>0) y = 1;
else {y = 2;  S};   //S does not write z
z = y;
```

- Which of the following statements about *z* are valid from the perspective of a static analysis?
  - The value of z is 1

A Simple Example

```
read(x);
if(x>0) y = 1;
else {y = 2;  S};   //S does not write z
z = y;
```

- Which of the following statements about *z* are valid from the perspective of a static analysis?
  - The value of z is 1

A Simple Example

```
read(x);
if(x>0) y = 1;
else {y = 2;  S};   //S does not write z
z = y;
```

- Which of the following statements about *z* are valid from the perspective of a static analysis?
  - The value of z is 1
  - The value of z is 2

A Simple Example

```
read(x);
if(x>0) y = 1;
else {y = 2;  S};   //S does not write z
z = y;
```

- Which of the following statements about *z* are valid from the perspective of a static analysis?
  - The value of z is 1
  - The value of z is 2

A Simple Example

```
read(x);
if(x>0) y = 1;
else {y = 2;  S};   //S does not write z
z = y;
```

- Which of the following statements about *z* are valid from the perspective of a static analysis?
  - The value of z is 1
  - The value of z is 2
  - The value of z is in the set {1, 2}

A Simple Example

```
read(x);
if(x>0) y = 1;
else {y = 2;  S};   //S does not write z
z = y;
```

- Which of the following statements about *z* are valid from the perspective of a static analysis?
  - The value of z is 1
  - The value of z is 2
  - The value of z is in the set {1, 2}

A Simple Example

```
read(x);
if(x>0) y = 1;
else {y = 2;  S};   //S does not write z
z = y;
```

- Which of the following statements about *z* are valid from the perspective of a static analysis?
  - The value of z is 1 ✗
  - The value of z is 2 ✗
  - The value of z is in the set {1, 2} √
  - The value of z is in the set {1, 2, 34, 128}

A Simple Example

```
read(x);
if(x>0) y = 1;
else {y = 2;  S};   //S does not write z
z = y;
```

- Which of the following statements about *z* are valid from the perspective of a static analysis?
  - The value of z is 1 ✗
  - The value of z is 2 ✗
  - The value of z is in the set {1, 2} √
  - The value of z is in the set {1, 2, 34, 128} √

A Simple Example

```
read(x);
if(x>0) y = 1;
else {y = 2;  S};   //S does not write z
z = y;
```

- Which of the following statements about *z* are valid from the perspective of a static analysis?
  - The value of z is 1
  - The value of z is 2
  - The value of z is in the set {1, 2}
  - The value of z is in the set {1, 2, 34, 128}
  - The value of z depends on the value of x; when x > 0, z is 1; otherwise z is 2

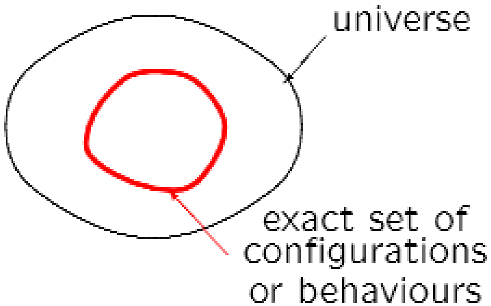
A Simple Example

```
read(x);
if(x>0) y = 1;
else {y = 2;  S};   //S does not write z
z = y;
```

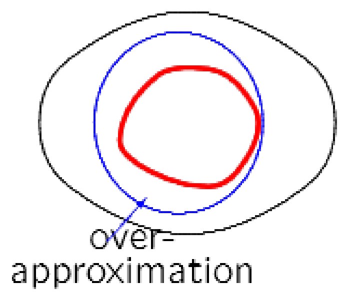
- Which of the following statements about *z* are valid from the perspective of a static analysis?
  - The value of z is 1
  - The value of z is 2
  - The value of z is in the set {1, 2}
  - The value of z is in the set {1, 2, 34, 128}
  - The value of z depends on the value of x; when x > 0, z is 1; otherwise z is 2

The Nature of Approximations

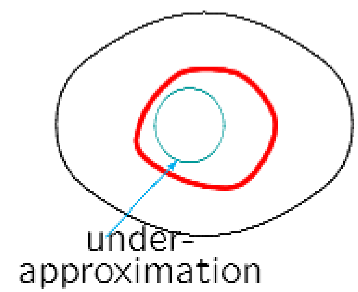
The exact world



Over-approximation



Under-approximation



Slogans: Err on the safe side!
Trade precision for efficiency!

Setting the Stage

- Formalism
  - A simple imperative language
  - Operational semantics
  - Lattice theory
  - Fixedpoint computation
- A simple reaching-definition analysis used throughout the quarter

A *while* Language

$$
\begin{array}{rcll}
a & \in & \mathbf{AExp} & \text{arithmetic expressions} \\
b & \in & \mathbf{BExp} & \text{boolean expressions} \\
S & \in & \mathbf{Stmt} & \text{statements} \\
x, y & \in & \mathbf{Var} & \text{variables} \\
n & \in & \mathbf{Num} & \text{numerals} \\
\ell & \in & \mathbf{Lab} & \text{labels} \\
op_a & \in & \mathbf{Op}_a & \text{arithmetic operators} \\
op_b & \in & \mathbf{Op}_b & \text{boolean operators} \\
op_r & \in & \mathbf{Op}_r & \text{relational operators}
\end{array}
$$

$$
\begin{array}{rcl}
a & ::= & x \mid n \mid a_1\ op_a\ a_2 \\
b & ::= & \mathtt{true} \mid \mathtt{false} \mid \mathtt{not}\ b \mid b_1\ op_b\ b_2 \mid a_1\ op_r\ a_2 \\
S & ::= & [x := a]^\ell \mid [\mathtt{skip}]^\ell \mid S_1; S_2 \mid \\
 & & \mathtt{if}\ [b]^\ell\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2 \mid \mathtt{while}\ [b]^\ell\ \mathtt{do}\ S
\end{array}
$$

An Example Program

$[y:=x]^1$; $[z:=1]^2$;

while $[y>1]^3$ do

$\quad$ ($[z:=z*y]^4$; $[y:=y-1]^5$;);

$[y:=0]^6$

Computes the factorial of the number in x and leaves the result in z

Formal Semantics

- Why useful
  - Formally define what a program does exactly
  - Prove the correctness of an language implementation or a program analysis
- Three major kinds of semantics
  - Denotational semantics
  - Operational semantics
  - Axiomatic semantics

Denotational Semantics

- Concerned about the conceptual meaning of a program
- Each phrase is interpreted as a denotation
- The meaning of a program reduces to the meaning of the sequence of commands

An Denotational Semantics Example

Syntactic Domains

N : Numeral -- nonnegative numerals

D : Digit -- decimal digits

Abstract Production Rules

Numeral ::= Digit | Numeral Digit

Digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Semantic Domain

Number = {0, 1, 2, 3, 4, … } -- natural numbers

Semantic Functions

$value$: Numeral $\rightarrow$ Number

$digit$: Digit $\rightarrow$ Number

Semantic Equations

$value\ [\![\mathrm{N\ D}]\!] = plus\ (times(10,\ value\ [\![\mathrm{N}]\!]),\ digit\ [\![\mathrm{D}]\!])$

$value\ [\![\mathrm{D}]\!] = digit\ [\![\mathrm{D}]\!]$

$digit\ [\![0]\!] = 0$	$digit\ [\![3]\!] = 3$	$digit\ [\![6]\!] = 6$	$digit\ [\![8]\!] = 8$
$digit\ [\![1]\!] = 1$	$digit\ [\![4]\!] = 4$	$digit\ [\![7]\!] = 7$	$digit\ [\![9]\!] = 9$
$digit\ [\![2]\!] = 2$	$digit\ [\![5]\!] = 5$		

Denotational Semantics

value ⟦1023⟧ = plus(times(10, value⟦102⟧), digit⟦3⟧)
= plus(times(10, plus(times(10,
value⟦10⟧, digit⟦2⟧))), digit⟦3⟧)
= plus(times(10, plus(times(10,
plus(times(10, plus(times(10, digit⟦1⟧), digit⟦0⟧))),
digit⟦2⟧))),digit⟦3⟧)
= 1023

Two language constructs are semantically equivalent if they share the same denotation

Axiomatic Semantics

- Based on mathematical logic (e.g., Hoare logic)
  - Used to reason about the correctness of a program
- Hoare triple
  - {P} C {Q}
  - P and Q are *assertions* (i.e., formulae in predicate logic) and C is a command
  - P is the precondition and Q is the postcondition
  - When P is met, C establishes Q
- Example: $\{x + 1 = 43\}\ y := x+1\ \{y = 43\}$

Operational Semantics

- The execution of a program is described directly
- Structural (small-step) operational semantics
  - Formally define how the individual steps of a computation take place
- Big-step operational semantics
  - How the overall results of an execution are obtained

Operational Semantics

- More commonly used in formally reasoning about a program analysis algorithm
  - The algorithm is sound if it appropriately abstracts the concrete operational semantics of the program

Operational Semantics

A *state* is a mapping from variables to integers:

$$\sigma \in \mathbf{State} \;=\; \mathbf{Var} \to \mathbf{Z}$$

The semantics of arithmetic and boolean expressions

$$\mathcal{A} : \quad \mathbf{AExp} \to (\mathbf{State} \to \mathbf{Z}) \quad \text{(no errors allowed)}$$

$$\mathcal{B} : \quad \mathbf{BExp} \to (\mathbf{State} \to \mathbf{T}) \quad \text{(no errors allowed)}$$

The *transitions* of the semantics are of the form

$$\langle S, \sigma \rangle \to \sigma' \quad \text{and} \quad \langle S, \sigma \rangle \to \langle S', \sigma' \rangle$$

Transitions

$$\langle [x := a]^\ell, \sigma\rangle \to \sigma[x \mapsto \mathcal{A}[\![a]\!]\sigma]$$

$$\langle [\mathtt{skip}]^\ell, \sigma\rangle \to \sigma$$

$$\frac{\langle S_1, \sigma\rangle \to \langle S_1', \sigma'\rangle}{\langle S_1; S_2, \sigma\rangle \to \langle S_1'; S_2, \sigma'\rangle}$$

$$\frac{\langle S_1, \sigma\rangle \to \sigma'}{\langle S_1; S_2, \sigma\rangle \to \langle S_2, \sigma'\rangle}$$

$$\langle \mathtt{if}\ [b]^\ell\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2, \sigma\rangle \to \langle S_1, \sigma\rangle \quad \text{if } \mathcal{B}[\![b]\!]\sigma = \mathit{true}$$

$$\langle \mathtt{if}\ [b]^\ell\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2, \sigma\rangle \to \langle S_2, \sigma\rangle \quad \text{if } \mathcal{B}[\![b]\!]\sigma = \mathit{false}$$

$$\langle \mathtt{while}\ [b]^\ell\ \mathtt{do}\ S, \sigma\rangle \to \langle (S; \mathtt{while}\ [b]^\ell\ \mathtt{do}\ S), \sigma\rangle \quad \text{if } \mathcal{B}[\![b]\!]\sigma = \mathit{true}$$

$$\langle \mathtt{while}\ [b]^\ell\ \mathtt{do}\ S, \sigma\rangle \to \sigma \quad \text{if } \mathcal{B}[\![b]\!]\sigma = \mathit{false}$$

Example Derivation Sequence

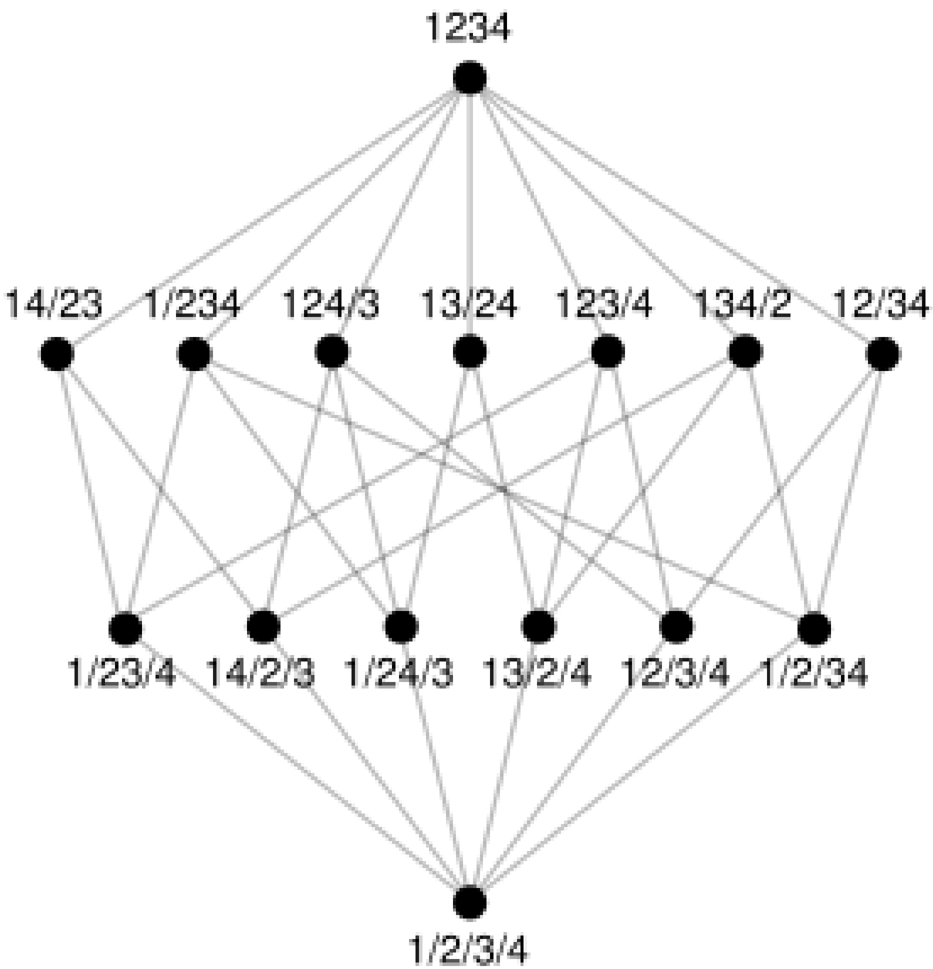
$$\langle [\texttt{y:=x}]^1; [\texttt{z:=1}]^2; \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{300}\rangle$$

$$\begin{aligned}
&\rightarrow \langle [\texttt{z:=1}]^2; \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{330}\rangle \\
&\rightarrow \langle \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{331}\rangle \\
&\rightarrow \langle [\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5; \\
&\qquad \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{331}\rangle \\
&\rightarrow \langle [\texttt{y:=y-1}]^5; \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{333}\rangle \\
&\rightarrow \langle \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{323}\rangle \\
&\rightarrow \langle [\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5; \\
&\qquad \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{323}\rangle \\
&\rightarrow \langle [\texttt{y:=y-1}]^5; \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{326}\rangle \\
&\rightarrow \langle \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{316}\rangle \\
&\rightarrow \langle [\texttt{y:=0}]^6, \sigma_{316}\rangle \\
&\rightarrow \sigma_{306}
\end{aligned}$$

Lattice Theory

- A lattice is a partially ordered set $(L, \leq)$
- Any two elements have a supremum (i.e., least upper bound) and an infimum (i.e., greatest lower bound)
- For any two elements a and b in L, a and b have a join: $a \vee b$ (superemum)
- For any two elements a and b in L, a and b have a meet: $a \wedge b$ (infimum)

An Example Lattice



- A lattice of partitions of a four-element set {1, 2, 3, 4}
- Ordered by the relation “is refinement of”
- a ∨ b = a coarser-grained partition than both a and b
- a ∧ b = a finer-grained partition than both a and b

General Properties

- Commutative laws
  - $a \land b = b \land a \quad a \lor b = b \lor a$
- Associative laws
  - $a \lor (b \lor c) = (a \lor b) \lor c \quad a \land (b \land c) = (a \land b) \land c$
- Absorption laws
  - $a \lor (a \land b) = a \quad a \land (a \lor b) = a$
- Idempotent laws
  - $a \lor a = a \quad a \land a = a$

More about Lattice

- The least element $\perp$ (i.e., unknown) and the greatest element $\top$ (i.e., everything)
  - $\top \wedge a = a \quad \top \vee a = \top$
  - $\perp \wedge a = \perp \quad \perp \vee a = a$
- Semi-lattice
  - A join-semi-lattice only has a join for any non-empty finite subset
  - A meet-semi-lattice only has a meet for any non-empty finite subset
- Real-world examples
  - Types in Java

Fixedpoint Computation

A fixedpoint equation has the form

$$f(x) = x$$

Its solutions are called the fixed points of f because if x_p is a solution then

$$x_p = f(x_p) = f(f(x_p)) = f(f(f(x_p))) = \ldots$$

In program analysis, we look for both such x_p and function f that can eventually reach a fixedpoint

Tarski's Fixedpoint Theorem

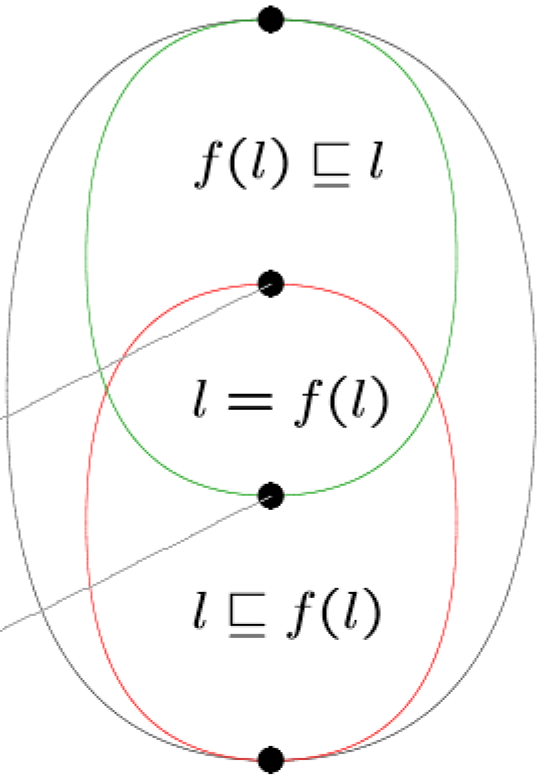
Let $L = (L, \sqsubseteq)$ be a complete lattice and let $f : L \to L$ be a monotone function.

The greatest fixed point $gfp(f)$ satisfy:

$$gfp(f) = \sqcup\{l \mid l \sqsubseteq f(l)\} \in \{l \mid f(l) = l\}$$

The least fixed point $lfp(f)$ satisfy:

$$lfp(f) = \sqcap\{l \mid f(l) \sqsubseteq l\} \in \{l \mid f(l) = l\}$$



Dataflow Analysis

Harry Xu

CS 253/INF 212

Spring 2013

Acknowledgements

Many slides in this file were taken from the chapter 2 slides available at

http://www2.imm.dtu.dk/~hrni/PPA/ppasup2004.html

We thank the authors of the book *Principles of Program Analysis* for providing their slides.

Dataflow analysis

- A class of static analyses that aim to understand how data flows in the program
- Typical examples
  - Available expression analysis
  - Reaching definition analysis
  - Live variable analysis
  - Constant propagation

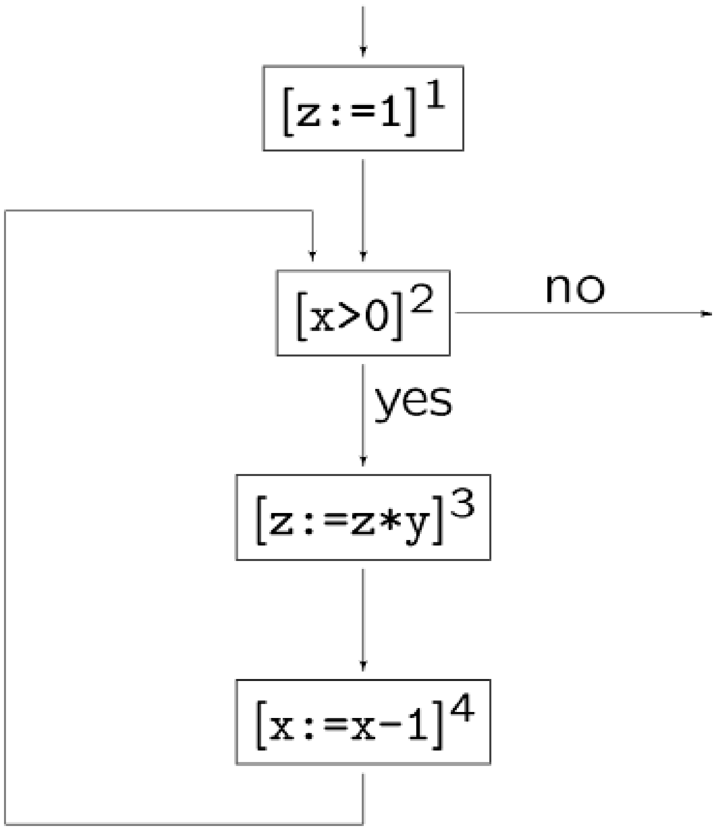
Analysis Scope

- Intraprocedural analysis
  - Focusing on each individual function
  - Do not track dataflow across function boundary
- Interprocedural analysis
  - Analyze the whole program
  - Way more expensive

Control flow graph

Example: $[\texttt{z:=1}]^1$; while $[\texttt{x>0}]^2$ do $([\texttt{z:=z*y}]^3; [\texttt{x:=x-1}]^4)$

$$\begin{aligned}
\mathit{init}(\cdots) &= 1 \\
\mathit{final}(\cdots) &= \{2\} \\
\mathit{labels}(\cdots) &= \{1,2,3,4\} \\
\mathit{flow}(\cdots) &= \{(1,2),(2,3), \\
&\quad (3,4),(4,2)\} \\
\mathit{flow}^R(\cdots) &= \{(2,1),(2,4), \\
&\quad (3,2),(4,3)\}
\end{aligned}$$



Intraprocedural Dataflow Analyses

- Classical analyses
  - Available expression analysis
  - Reaching definition analysis
  - Live variable analysis

Available Expression Analysis

The aim of the *Available Expressions Analysis* is to determine

> For each program point, which expressions must have already been computed, and not later modified, on all paths to the program point.

Example:

point of interest

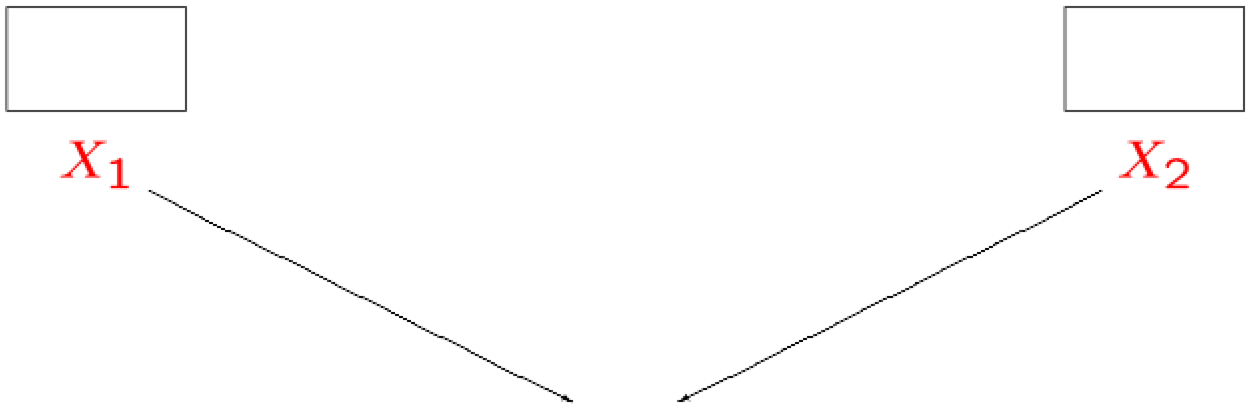
$$\Downarrow$$

$$[\texttt{x:=}\ \texttt{a+b}\ ]^1; [\texttt{y:=a*b}]^2; \texttt{while}\ [\texttt{y>}\ \texttt{a+b}\ ]^3\ \texttt{do}\ ([\texttt{a:=a+1}]^4; [\texttt{x:=}\ \texttt{a+b}\ ]^5)$$

The analysis enables a transformation into

$$[\texttt{x:= a+b}]^1; [\texttt{y:=a*b}]^2; \texttt{while}\ [\texttt{y>}\ \texttt{x}\ ]^3\ \texttt{do}\ ([\texttt{a:=a+1}]^4; [\texttt{x:= a+b}]^5)$$

Basic Idea



$$N = X_1 \cap X_2$$

$$\boxed{x := a}$$

$$X = (N \setminus \overbrace{\{\text{expressions with an } x\}}^{\text{kill}}) \cup \underbrace{\{\text{subexpressions of } a \text{ without an } x\}}_{\text{gen}}$$

Analysis Algorithm

kill and *gen* functions

$$
\begin{aligned}
kill_{\mathsf{AE}}([x := a]^{\ell}) &= \{a' \in \mathbf{AExp}_{\star} \mid x \in FV(a')\} \\
kill_{\mathsf{AE}}([\texttt{skip}]^{\ell}) &= \emptyset \\
kill_{\mathsf{AE}}([b]^{\ell}) &= \emptyset \\
\\
gen_{\mathsf{AE}}([x := a]^{\ell}) &= \{a' \in \mathbf{AExp}(a) \mid x \notin FV(a')\} \\
gen_{\mathsf{AE}}([\texttt{skip}]^{\ell}) &= \emptyset \\
gen_{\mathsf{AE}}([b]^{\ell}) &= \mathbf{AExp}(b)
\end{aligned}
$$

data flow equations: $\mathsf{AE}^{=}$

$$
\mathsf{AE}_{entry}(\ell) = \begin{cases} \emptyset & \text{if } \ell = init(S_{\star}) \\ \bigcap\{\mathsf{AE}_{exit}(\ell') \mid (\ell', \ell) \in flow(S_{\star})\} & \text{otherwise} \end{cases}
$$

$$
\mathsf{AE}_{exit}(\ell) = (\mathsf{AE}_{entry}(\ell) \backslash kill_{\mathsf{AE}}(B^{\ell})) \cup gen_{\mathsf{AE}}(B^{\ell})
$$

where $B^{\ell} \in blocks(S_{\star})$

Analysis Example

$$[\texttt{x:=a+b}]^1; [\texttt{y:=a*b}]^2; \texttt{while } [\texttt{y>a+b}]^3 \texttt{ do } ([\texttt{a:=a+1}]^4; [\texttt{x:=a+b}]^5)$$

kill and *gen* functions:

ℓ	$kill_{AE}(\ell)$	$gen_{AE}(\ell)$
1	$\emptyset$	{a+b}
2	$\emptyset$	{a*b}
3	$\emptyset$	{a+b}
4	{a+b, a*b, a+1}	$\emptyset$
5	$\emptyset$	{a+b}

Example (Cond)

$[\texttt{x:=a+b}]^1; [\texttt{y:=a*b}]^2; \texttt{while } [\texttt{y>a+b}]^3 \texttt{ do } ([\texttt{a:=a+1}]^4; [\texttt{x:=a+b}]^5)$

Equations:

$$
\begin{aligned}
\mathsf{AE}_{entry}(1) &= \emptyset \\
\mathsf{AE}_{entry}(2) &= \mathsf{AE}_{exit}(1) \\
\mathsf{AE}_{entry}(3) &= \mathsf{AE}_{exit}(2) \cap \mathsf{AE}_{exit}(5) \\
\mathsf{AE}_{entry}(4) &= \mathsf{AE}_{exit}(3) \\
\mathsf{AE}_{entry}(5) &= \mathsf{AE}_{exit}(4)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{AE}_{exit}(1) &= \mathsf{AE}_{entry}(1) \cup \{\texttt{a+b}\} \\
\mathsf{AE}_{exit}(2) &= \mathsf{AE}_{entry}(2) \cup \{\texttt{a*b}\} \\
\mathsf{AE}_{exit}(3) &= \mathsf{AE}_{entry}(3) \cup \{\texttt{a+b}\} \\
\mathsf{AE}_{exit}(4) &= \mathsf{AE}_{entry}(4) \backslash \{\texttt{a+b}, \texttt{a*b}, \texttt{a+1}\} \\
\mathsf{AE}_{exit}(5) &= \mathsf{AE}_{entry}(5) \cup \{\texttt{a+b}\}
\end{aligned}
$$

Example (Cond)

$$[\texttt{x:=a+b}]^1; [\texttt{y:=a*b}]^2; \texttt{while } [\texttt{y> a+b}]^3 \texttt{ do } ([\texttt{a:=a+1}]^4; [\texttt{x:=a+b}]^5)$$

Largest solution:

ℓ	$\mathsf{AE}_{entry}(\ell)$	$\mathsf{AE}_{exit}(\ell)$
1	$\emptyset$	{a+b}
2	{a+b}	{a+b, a*b}
3	{a+b}	{a+b}
4	{a+b}	$\emptyset$
5	$\emptyset$	{a+b}

Reaching Definition Analysis

The aim of the *Reaching Definitions Analysis* is to determine

> For each program point, which assignments may have been made and not overwritten, when program execution reaches this point along some path.

Example:

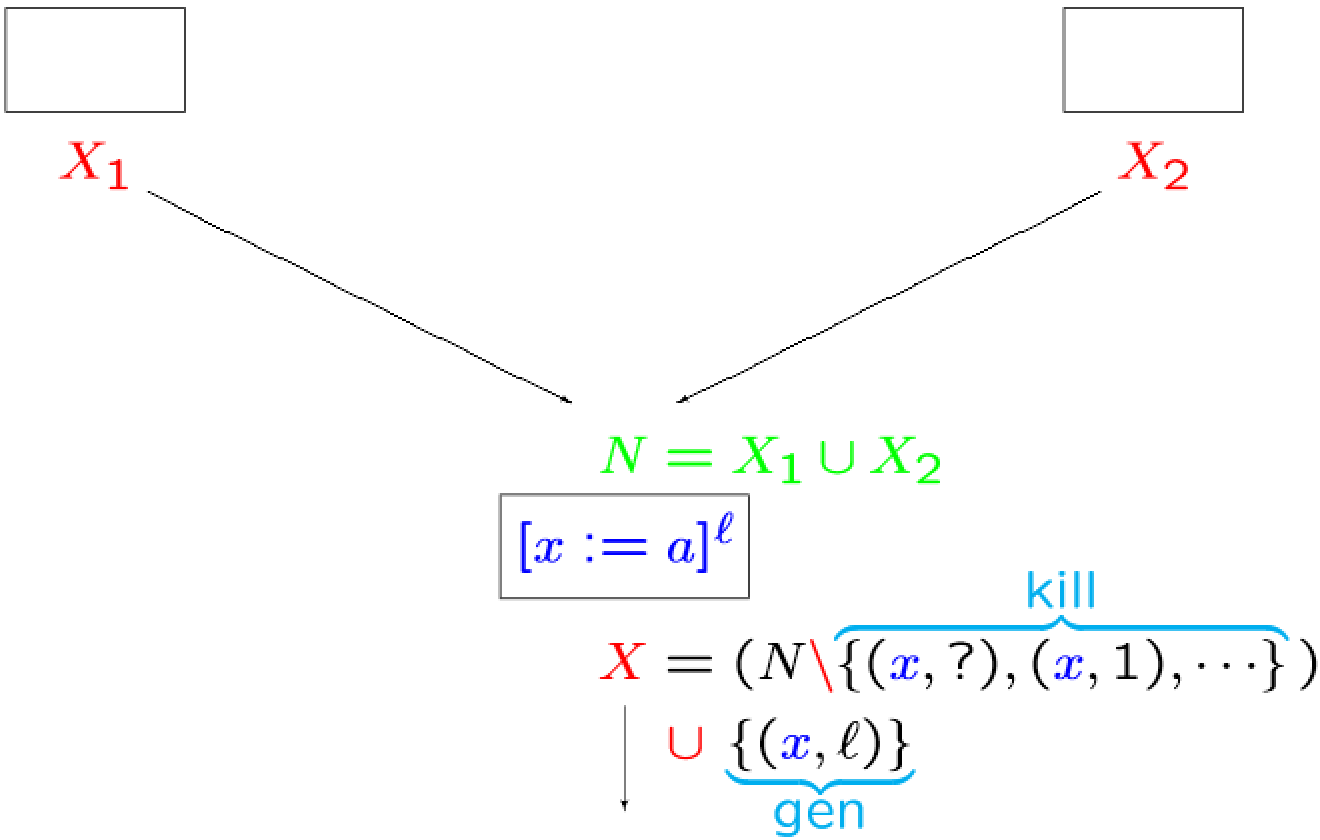
point of interest

$\Downarrow$

$$[\mathtt{x:=5}]^1; [\mathtt{y:=1}]^2; \mathtt{while}\ [\mathtt{x>1}]^3\ \mathtt{do}\ ([\mathtt{y:=x*y}]^4; [\mathtt{x:=x-1}]^5)$$

useful for definition-use chains and use-definition chains

Basic Idea



X_1
X_2
$N = X_1 \cup X_2$
$\boxed{[x := a]^\ell}$
$X = (N \setminus \overbrace{\{(x, ?), (x, 1), \cdots\}}^{\text{kill}})$
$\cup \underbrace{\{(x, \ell)\}}_{\text{gen}}$

Analysis Algorithm

kill and *gen* functions

$$
\begin{aligned}
\mathit{kill}_{\mathsf{RD}}([x := a]^\ell) &= \{(x, ?)\} \\
&\quad \cup \{(x, \ell') \mid B^{\ell'} \text{ is an assignment to } x \text{ in } S_\star\} \\
\mathit{kill}_{\mathsf{RD}}([\mathtt{skip}]^\ell) &= \emptyset \\
\mathit{kill}_{\mathsf{RD}}([b]^\ell) &= \emptyset \\
\mathit{gen}_{\mathsf{RD}}([x := a]^\ell) &= \{(x, \ell)\} \\
\mathit{gen}_{\mathsf{RD}}([\mathtt{skip}]^\ell) &= \emptyset \\
\mathit{gen}_{\mathsf{RD}}([b]^\ell) &= \emptyset
\end{aligned}
$$

data flow equations: $\mathsf{RD}^=$

$$
\mathsf{RD}_{entry}(\ell) = \begin{cases} \{(x, ?) \mid x \in \mathit{FV}(S_\star)\} & \text{if } \ell = \mathit{init}(S_\star) \\ \bigcup\{\mathsf{RD}_{exit}(\ell') \mid (\ell', \ell) \in \mathit{flow}(S_\star)\} & \text{otherwise} \end{cases}
$$

$$
\mathsf{RD}_{exit}(\ell) = (\mathsf{RD}_{entry}(\ell) \backslash \mathit{kill}_{\mathsf{RD}}(B^\ell)) \cup \mathit{gen}_{\mathsf{RD}}(B^\ell)
$$

where $B^\ell \in \mathit{blocks}(S_\star)$

Analysis Example

$$[\mathtt{x:=5}]^1; [\mathtt{y:=1}]^2; \mathtt{while}\ [\mathtt{x>1}]^3\ \mathtt{do}\ ([\mathtt{y:=x*y}]^4; [\mathtt{x:=x-1}]^5)$$

kill and *gen* functions:

ℓ	$\mathit{kill}_{\mathsf{RD}}(\ell)$	$\mathit{gen}_{\mathsf{RD}}(\ell)$
1	$\{(x,?),(x,1),(x,5)\}$	$\{(x,1)\}$
2	$\{(y,?),(y,2),(y,4)\}$	$\{(y,2)\}$
3	$\emptyset$	$\emptyset$
4	$\{(y,?),(y,2),(y,4)\}$	$\{(y,4)\}$
5	$\{(x,?),(x,1),(x,5)\}$	$\{(x,5)\}$

Example (Cond)

$$[\mathtt{x:=5}]^1; [\mathtt{y:=1}]^2; \mathtt{while}\ [\mathtt{x>1}]^3\ \mathtt{do}\ ([\mathtt{y:=x*y}]^4; [\mathtt{x:=x-1}]^5)$$

Equations:

$$\begin{aligned}
\mathsf{RD}_{entry}(1) &= \{(x,?),(y,?)\}\\
\mathsf{RD}_{entry}(2) &= \mathsf{RD}_{exit}(1)\\
\mathsf{RD}_{entry}(3) &= \mathsf{RD}_{exit}(2) \cup \mathsf{RD}_{exit}(5)\\
\mathsf{RD}_{entry}(4) &= \mathsf{RD}_{exit}(3)\\
\mathsf{RD}_{entry}(5) &= \mathsf{RD}_{exit}(4)
\end{aligned}$$

$$\begin{aligned}
\mathsf{RD}_{exit}(1) &= (\mathsf{RD}_{entry}(1)\backslash\{(x,?),(x,1),(x,5)\}) \cup \{(x,1)\}\\
\mathsf{RD}_{exit}(2) &= (\mathsf{RD}_{entry}(2)\backslash\{(y,?),(y,2),(y,4)\}) \cup \{(y,2)\}\\
\mathsf{RD}_{exit}(3) &= \mathsf{RD}_{entry}(3)\\
\mathsf{RD}_{exit}(4) &= (\mathsf{RD}_{entry}(4)\backslash\{(y,?),(y,2),(y,4)\}) \cup \{(y,4)\}\\
\mathsf{RD}_{exit}(5) &= (\mathsf{RD}_{entry}(5)\backslash\{(x,?),(x,1),(x,5)\}) \cup \{(x,5)\}
\end{aligned}$$

Example (Cond)

$[\mathtt{x:=5}]^1; [\mathtt{y:=1}]^2; \mathtt{while}\ [\mathtt{x>1}]^3\ \mathtt{do}\ ([\mathtt{y:=x*y}]^4; [\mathtt{x:=x-1}]^5)$

Smallest solution:

ℓ	$\mathsf{RD}_{entry}(\ell)$	$\mathsf{RD}_{exit}(\ell)$
1	$\{(x,?),(y,?)\}$	$\{(y,?),(x,1)\}$
2	$\{(y,?),(x,1)\}$	$\{(x,1),(y,2)\}$
3	$\{(x,1),(y,2),(y,4),(x,5)\}$	$\{(x,1),(y,2),(y,4),(x,5)\}$
4	$\{(x,1),(y,2),(y,4),(x,5)\}$	$\{(x,1),(y,4),(x,5)\}$
5	$\{(x,1),(y,4),(x,5)\}$	$\{(y,4),(x,5)\}$

Live Variable Analysis

A variable is *live* at the exit from a label if there is a path from the label to a use of the variable that does not re-define the variable.

The aim of the *Live Variables Analysis* is to determine

> For each program point, which variables may be live at the exit from the point.

Example:

point of interest

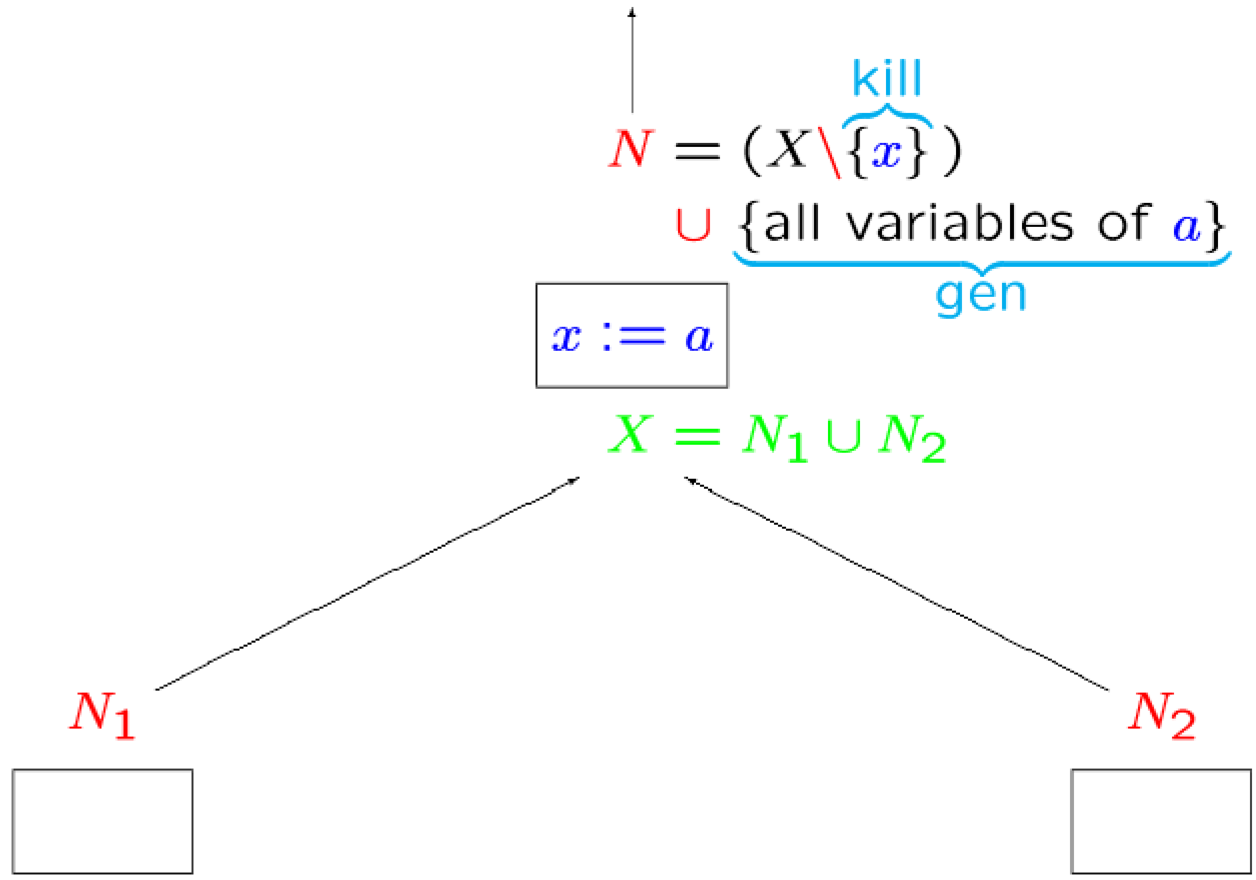
$\Downarrow$

$[\texttt{x}:=2]^1; [\texttt{y}:=4]^2; [\texttt{x}:=1]^3; (\texttt{if } [\texttt{y}>\texttt{x}]^4 \texttt{ then } [\texttt{z}:=\texttt{y}]^5 \texttt{ else } [\texttt{z}:=\texttt{y}*\texttt{y}]^6); [\texttt{x}:=\texttt{z}]^7$

The analysis enables a transformation into

$$[\texttt{y}:=4]^2; [\texttt{x}:=1]^3; (\texttt{if } [\texttt{y}>\texttt{x}]^4 \texttt{ then } [\texttt{z}:=\texttt{y}]^5 \texttt{ else } [\texttt{z}:=\texttt{y}*\texttt{y}]^6); [\texttt{x}:=\texttt{z}]^7$$

Basic Idea


$N = (X \setminus \overbrace{\{x\}}^{\text{kill}})$
$\cup \underbrace{\{\text{all variables of } a\}}_{\text{gen}}$
$x := a$
$X = N_1 \cup N_2$
N_1
N_2

Analysis Algorithm

kill and *gen* functions

$$
\begin{aligned}
\mathit{kill}_{\mathsf{LV}}([x := a]^\ell) &= \{x\} \\
\mathit{kill}_{\mathsf{LV}}([\mathtt{skip}]^\ell) &= \emptyset \\
\mathit{kill}_{\mathsf{LV}}([b]^\ell) &= \emptyset
\end{aligned}
$$

$$
\begin{aligned}
\mathit{gen}_{\mathsf{LV}}([x := a]^\ell) &= \mathit{FV}(a) \\
\mathit{gen}_{\mathsf{LV}}([\mathtt{skip}]^\ell) &= \emptyset \\
\mathit{gen}_{\mathsf{LV}}([b]^\ell) &= \mathit{FV}(b)
\end{aligned}
$$

data flow equations: $\mathsf{LV}^=$

$$
\mathsf{LV}_{exit}(\ell) = \begin{cases} \emptyset & \text{if } \ell \in \mathit{final}(S_\star) \\ \bigcup\{\mathsf{LV}_{entry}(\ell') \mid (\ell', \ell) \in \mathit{flow}^R(S_\star)\} & \text{otherwise} \end{cases}
$$

$$
\mathsf{LV}_{entry}(\ell) = (\mathsf{LV}_{exit}(\ell) \backslash \mathit{kill}_{\mathsf{LV}}(B^\ell)) \cup \mathit{gen}_{\mathsf{LV}}(B^\ell)
$$

where $B^\ell \in \mathit{blocks}(S_\star)$

Example

$$[\mathtt{x:=2}]^1; [\mathtt{y:=4}]^2; [\mathtt{x:=1}]^3; (\mathtt{if}\ [\mathtt{y>x}]^4\ \mathtt{then}\ [\mathtt{z:=y}]^5\ \mathtt{else}\ [\mathtt{z:=y*y}]^6); [\mathtt{x:=z}]^7$$

kill and *gen* functions:

ℓ	$\mathit{kill}_{\mathsf{LV}}(\ell)$	$\mathit{gen}_{\mathsf{LV}}(\ell)$
1	$\{\mathtt{x}\}$	$\emptyset$
2	$\{\mathtt{y}\}$	$\emptyset$
3	$\{\mathtt{x}\}$	$\emptyset$
4	$\emptyset$	$\{\mathtt{x}, \mathtt{y}\}$
5	$\{\mathtt{z}\}$	$\{\mathtt{y}\}$
6	$\{\mathtt{z}\}$	$\{\mathtt{y}\}$
7	$\{\mathtt{x}\}$	$\{\mathtt{z}\}$

Example (Cond)

$[\texttt{x:=2}]^1; [\texttt{y:=4}]^2; [\texttt{x:=1}]^3; (\texttt{if } [\texttt{y>x}]^4 \texttt{ then } [\texttt{z:=y}]^5 \texttt{ else } [\texttt{z:=y*y}]^6); [\texttt{x:=z}]^7$

Equations:

$$
\begin{aligned}
\mathsf{LV}_{entry}(1) &= \mathsf{LV}_{exit}(1)\backslash\{\texttt{x}\} \\
\mathsf{LV}_{entry}(2) &= \mathsf{LV}_{exit}(2)\backslash\{\texttt{y}\} \\
\mathsf{LV}_{entry}(3) &= \mathsf{LV}_{exit}(3)\backslash\{\texttt{x}\} \\
\mathsf{LV}_{entry}(4) &= \mathsf{LV}_{exit}(4) \cup \{\texttt{x},\texttt{y}\} \\
\mathsf{LV}_{entry}(5) &= (\mathsf{LV}_{exit}(5)\backslash\{\texttt{z}\}) \cup \{\texttt{y}\} \\
\mathsf{LV}_{entry}(6) &= (\mathsf{LV}_{exit}(6)\backslash\{\texttt{z}\}) \cup \{\texttt{y}\} \\
\mathsf{LV}_{entry}(7) &= \{\texttt{z}\}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{LV}_{exit}(1) &= \mathsf{LV}_{entry}(2) \\
\mathsf{LV}_{exit}(2) &= \mathsf{LV}_{entry}(3) \\
\mathsf{LV}_{exit}(3) &= \mathsf{LV}_{entry}(4) \\
\mathsf{LV}_{exit}(4) &= \mathsf{LV}_{entry}(5) \cup \mathsf{LV}_{entry}(6) \\
\mathsf{LV}_{exit}(5) &= \mathsf{LV}_{entry}(7) \\
\mathsf{LV}_{exit}(6) &= \mathsf{LV}_{entry}(7) \\
\mathsf{LV}_{exit}(7) &= \emptyset
\end{aligned}
$$

Example (Cond)

$[\texttt{x:=2}]^1; [\texttt{y:=4}]^2; [\texttt{x:=1}]^3; (\texttt{if } [\texttt{y>x}]^4 \texttt{ then } [\texttt{z:=y}]^5 \texttt{ else } [\texttt{z:=y*y}]^6); [\texttt{x:=z}]^7$

Smallest solution:

ℓ	$\mathsf{LV}_{entry}(\ell)$	$\mathsf{LV}_{exit}(\ell)$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{\texttt{y}\}$
3	$\{\texttt{y}\}$	$\{\texttt{x}, \texttt{y}\}$
4	$\{\texttt{x}, \texttt{y}\}$	$\{\texttt{y}\}$
5	$\{\texttt{y}\}$	$\{\texttt{z}\}$
6	$\{\texttt{y}\}$	$\{\texttt{z}\}$
7	$\{\texttt{z}\}$	$\emptyset$

Extracting Similarities

A common pattern exists in these analyses

$$\mathit{Analysis}_\circ(\ell) = \begin{cases} \iota & \text{if } \ell \in E \\ \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell', \ell) \in F\} & \text{otherwise} \end{cases}$$

$$\mathit{Analysis}_\bullet(\ell) = f_\ell(\mathit{Analysis}_\circ(\ell))$$

where

- $\bigsqcup$ is $\bigcap$ or $\bigcup$ (and $\sqcup$ is $\cup$ or $\cap$),
- F is either $\mathit{flow}(S_\star)$ or $\mathit{flow}^R(S_\star)$,
- E is $\{\mathit{init}(S_\star)\}$ or $\mathit{final}(S_\star)$,
- ι specifies the initial or final analysis information, and
- f_ℓ is the transfer function associated with $B^\ell \in \mathit{blocks}(S_\star)$.

Forward v.s. Backward

- The *forward analyses* have F to be *flow*$(S_\star)$ and then *Analysis*$_\circ$ concerns entry conditions and *Analysis*$_\bullet$ concerns exit conditions; the equation system presupposes that $S_\star$ has isolated entries.

- The *backward analyses* have F to be *flow*$^R(S_\star)$ and then *Analysis*$_\circ$ concerns exit conditions and *Analysis*$_\bullet$ concerns entry conditions; the equation system presupposes that $S_\star$ has isolated exits.

Union or Intersection

- When $\bigsqcup$ is $\bigcap$ we require the greatest sets that solve the equations and we are able to detect properties satisfied by *all execution paths* reaching (or leaving) the entry (or exit) of a label; the analysis is called a must-analysis.

- When $\bigsqcup$ is $\bigcup$ we require the smallest sets that solve the equations and we are able to detect properties satisfied by *at least one execution path* to (or from) the entry (or exit) of a label; the analysis is called a may-analysis.

Property Space

L is a complete lattice used to represent the data flow information (data flow facts)

$\sqcup$ is the combination operation: $P(L) \longrightarrow L$, used to Combine information from different paths

Transfer Function

The set of transfer functions, $\mathcal{F}$, is a set of monotone functions over L, meaning that

$$l \sqsubseteq l' \text{ implies } f_\ell(l) \sqsubseteq f_\ell(l')$$

and furthermore they fulfil the following conditions:

- $\mathcal{F}$ contains *all* the transfer functions $f_\ell : L \to L$ in question (for $\ell \in \mathbf{Lab}_\star$)
- $\mathcal{F}$ contains the *identity function*
- $\mathcal{F}$ is *closed under composition* of functions

Frameworks

A *Monotone Framework* consists of:

- a complete lattice, L, that satisfies the Ascending Chain Condition; we write $\bigsqcup$ for the least upper bound operator
- a set $\mathcal{F}$ of monotone functions from L to L that contains the identity function and that is closed under function composition

A *Distributive Framework* is a Monotone Framework where additionally all functions f in $\mathcal{F}$ are required to be distributive:

$$f(l_1 \sqcup l_2) = f(l_1) \sqcup f(l_2)$$

Framework Instances

An *instance* of a Framework consists of:

– the complete lattice, L, of the framework

– the space of functions, $\mathcal{F}$, of the framework

– a finite flow, F (typically *flow*$(S_\star)$ or *flow*$^R(S_\star)$)

– a finite set of *extremal labels*, E (typically $\{$*init*$(S_\star)\}$ or *final*$(S_\star)$)

– an *extremal value*, $\iota \in L$, for the extremal labels

– a mapping, $f_\cdot$, from the labels $\mathbf{Lab}_\star$ to transfer functions in $\mathcal{F}$

Equations and Constraints

Equations of the Instance:

$$\begin{aligned} \mathit{Analysis}_\circ(\ell) &= \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell',\ell) \in F\} \sqcup \iota_E^\ell \\ &\qquad \text{where } \iota_E^\ell = \begin{cases} \iota & \text{if } \ell \in E \\ \bot & \text{if } \ell \notin E \end{cases} \\ \mathit{Analysis}_\bullet(\ell) &= f_\ell(\mathit{Analysis}_\circ(\ell)) \end{aligned}$$

Constraints of the Instance:

$$\begin{aligned} \mathit{Analysis}_\circ(\ell) &\sqsupseteq \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell',\ell) \in F\} \sqcup \iota_E^\ell \\ &\qquad \text{where } \iota_E^\ell = \begin{cases} \iota & \text{if } \ell \in E \\ \bot & \text{if } \ell \notin E \end{cases} \\ \mathit{Analysis}_\bullet(\ell) &\sqsupseteq f_\ell(\mathit{Analysis}_\circ(\ell)) \end{aligned}$$

Examples Revisited

	Available Expressions	Reaching Definitions	Very Busy Expressions	Live Variables
L	$\mathcal{P}(\mathbf{AExp}_\star)$	$\mathcal{P}(\mathbf{Var}_\star \times \mathbf{Lab}_\star)$	$\mathcal{P}(\mathbf{AExp}_\star)$	$\mathcal{P}(\mathbf{Var}_\star)$
$\sqsubseteq$	$\supseteq$	$\subseteq$	$\supseteq$	$\subseteq$
$\bigsqcup$	$\bigcap$	$\bigcup$	$\bigcap$	$\bigcup$
$\bot$	$\mathbf{AExp}_\star$	$\emptyset$	$\mathbf{AExp}_\star$	$\emptyset$
ι	$\emptyset$	$\{(x,?) \mid x \in \mathit{FV}(S_\star)\}$	$\emptyset$	$\emptyset$
E	$\{\mathit{init}(S_\star)\}$	$\{\mathit{init}(S_\star)\}$	$\mathit{final}(S_\star)$	$\mathit{final}(S_\star)$
F	$\mathit{flow}(S_\star)$	$\mathit{flow}(S_\star)$	$\mathit{flow}^R(S_\star)$	$\mathit{flow}^R(S_\star)$
$\mathcal{F}$	$\{f : L \to L \mid \exists l_k, l_g : f(l) = (l \setminus l_k) \cup l_g\}$			
f_ℓ	$f_\ell(l) = (l \setminus \mathit{kill}(B^\ell)) \cup \mathit{gen}(B^\ell)$ where $B^\ell \in \mathit{blocks}(S_\star)$			

Bit-Vector Frameworks

A *Bit Vector Framework* has

- $L = \mathcal{P}(D)$ for D finite
- $\mathcal{F} = \{f \mid \exists l_k, l_g : f(l) = (l \setminus l_k) \cup l_g\}$

Examples:

- Available Expressions
- Live Variables
- Reaching Definitions
- Very Busy Expressions

Bit-Vector Frameworks are Monotone and Distributive

$$
\begin{aligned}
f(l_1 \sqcup l_2) &= \begin{cases} f(l_1 \cup l_2) \\ f(l_1 \cap l_2) \end{cases} &&= \begin{cases} ((l_1 \cup l_2) \setminus l_k) \cup l_g \\ ((l_1 \cap l_2) \setminus l_k) \cup l_g \end{cases} \\
&= \begin{cases} ((l_1 \setminus l_k) \cup (l_2 \setminus l_k)) \cup l_g \\ ((l_1 \setminus l_k) \cap (l_2 \setminus l_k)) \cup l_g \end{cases} &&= \begin{cases} ((l_1 \setminus l_k) \cup l_g) \cup ((l_2 \setminus l_k) \cup l_g) \\ ((l_1 \setminus l_k) \cup l_g) \cap ((l_2 \setminus l_k) \cup l_g) \end{cases} \\
&= \begin{cases} f(l_1) \cup f(l_2) \\ f(l_1) \cap f(l_2) \end{cases} &&= f(l_1) \sqcup f(l_2)
\end{aligned}
$$

Monotonicity can be proved in a similar manner

Example: Constant Propagation

- Determine, for each program point, whether or not a variable has a constant value whenever execution reaches the point

Example:

$$[x:=6]^1; [y:=3]^2; \texttt{while } [x > \boxed{y}]^3 \texttt{ do } ([x:=x-1]^4; [z:=\boxed{y * y}]^6)$$

The analysis enables a transformation into

$$[x:=6]^1; [y:=3]^2; \texttt{while } [x > 3]^3 \texttt{ do } ([x:=x-1]^4; [z:=9]^6)$$

Now You Tell Me

- How to define a lattice L?
- How to define transfer functions?
- Is constant propagation a monotone framework?
- Is it a distributive framework?

Solving the Equation

- Many different approaches
- The least fixed-point solution
  - Always decidable
  - A worklist-based algorithm for monotone frameworks

Algorithm

- Idea: iterate until stabilization

Worklist Algorithm

Input: An instance $(L, \mathcal{F}, F, E, \iota, f_\cdot)$ of a Monotone Framework

Output: The MFP Solution: $MFP_\circ, MFP_\bullet$

Data structures:

- Analysis: the current analysis result for block entries (or exits)
- The worklist W: a list of pairs (ℓ, ℓ') indicating that the current analysis result has changed at the entry (or exit) to the block ℓ and hence the entry (or exit) information must be recomputed for ℓ'

Algorithm (Cond.)

Step 1 Initialisation (of W and Analysis)

W := nil;
for all (ℓ, ℓ') in F do W := cons$((\ell, \ell')$,W);
for all ℓ in F or E do
  if $\ell \in E$ then Analysis$[\ell]$:= ι else Analysis$[\ell]$:= $\perp_L$;

Step 2 Iteration (updating W and Analysis)

while W $\neq$ nil do
  ℓ := fst(head(W)); ℓ' = snd(head(W)); W := tail(W);
  if f_ℓ(Analysis$[\ell]$) $\not\sqsubseteq$ Analysis$[\ell']$ then
    Analysis$[\ell']$:= Analysis$[\ell']$ $\sqcup$ f_ℓ(Analysis$[\ell]$);
    for all ℓ'' with (ℓ', ℓ'') in F do W := cons$((\ell', \ell'')$,W);

Step 3 Presenting the result ($MFP_\circ$ and $MFP_\bullet$)

for all ℓ in F or E do
    $MFP_\circ(\ell)$:= Analysis$[\ell]$;
    $MFP_\bullet(\ell)$:= f_ℓ(Analysis$[\ell]$)