DATA FLOW ANALYSIS (INTRAPROCEDURAL)

Neil D. Jones
DIKU, University of Copenhagen (prof. emeritus)

COURSE MATERIAL

Book: NNH = Nielson, Nielson and Hankin **Principles of Program Analysis**

Slides: downloadable from course home page.

Reading for lectures 2 and 7 December:

- Read and understand NNH sections 1.1, 1.2, 1.3, 1.7, 1.8.
- Skim 1.4, 1.5, 1.6.
- Read and understand NNH section 2.1.

The **compiler construction course project** may have some application-oriented work based on Chapters 1 and 2.

SOURCES

How is program analysis done?

- Many people: **decades of practical experience** in writing compilers
  (though correctness issues are rarely addressed by compiler hackers)
- Engineering methodology: **program analysis by fix-point computations**.
- This was developed by informal, pragmatic, ad hoc methods from the 1950s called **data flow analysis**.

Semantics-based program analysis:

- Methods formally based in program semantics developed by Cousot+Cousot, Jones, Muchnick, Nielson+Nielson, Hankin, many others.
- Research since 1970's under the name of **Abstract Interpretation**
- Capture a significant part of data flow analysis (but not all).
- January 2008 conference in San Francisco:
  "30 Years of Abstract Interpretation."

MOTIVATION, ORIGINS

Optimising transformations for compilers.

Compiler structure:

$$\text{sourcecode} \longrightarrow \text{intermediatecode} \longrightarrow \text{intermediatecode} \longrightarrow \text{targetcode}$$

The Optimisation phase:

$$\text{intermediatecode} \longrightarrow \text{intermediatecode}$$

Intermediate code is usually (some version of) simple flow chart programs. These contain

- program points (also called labels),
- with an elementary statement or test at each point, and
- control transitions from one program point to another.

WHAT AND HOW

What: program transformation to improve efficiency

- Based on program flow analysis
- Must be correct (and just what does this mean?)
- Complex
- Important: efficiency, complex hardware, limits to what humans can improve, etc

How: several steps in program optimisation. First: program analysis.

- Choose a data flow lattice to describe program properties
- Build a system of data flow equations from the program
- Solve the system of data flow equations

Then transform the program, usually to optimise it

TOWARDS UNDERSTANDING THE PROBLEM I

Consider a transformation

$$[\mathtt{x := a}]^\ell \Rightarrow [\mathtt{skip}]^\ell$$

to eliminate code. (It sounds trivial, but it's significant in practice!)

Some possible reasons it can be correct:

1. Point ℓ is unreachable: control cannot flow from the program's start to $[\mathtt{x := a}]^\ell$
2. Point ℓ is dead: control cannot flow from $[\mathtt{x := a}]^\ell$ to the program's exit. For example
   - The program will definitely loop after point ℓ. Or
   - The program will definitely abort execution after point ℓ.
3. Variable $\mathtt{x}$ is dead at ℓ (even though point ℓ is not dead): For instance
   - $\mathtt{x}$ is never referenced again; or
   - $\mathtt{x}$ may be used to compute $\mathtt{y}$, $\mathtt{z}$, ... but they are never used again, ...

TOWARDS UNDERSTANDING THE PROBLEM II

More possible reasons for correctness of the transformation

$$[\texttt{x := a}]^\ell \Rightarrow [\texttt{skip}]^\ell$$

to eliminate code.

4. x is already equal to a (if control ever gets to ℓ)

5. Mathematical reasons relating x and a, e.g., Matiyasevich's theorem etc.

6. a is an uninitialised variable: so the value of x is completely undependable

7. Some patchwork combination of the above.
   (Eg, reason 3 applies if x is even, reason 4 applies if x is odd,. . .)

ALAS, MOST OF THESE REASONS ARE AS UNDECIDABLE AS THE HALTING PROBLEM (!)

Remark: many (most!) of the above program behavior properties are **undecidable** (if you insist on exact answers).

Proof See Rice's Theorem from Computability Theory.

So what do we do?

- The practice of **program analysis** and the theory of **abstract interpretation**: find **safe** descriptions of program behavior. Meaning of safety:
  - if the analysis says that a program has a certain behavior (e.g., that `x` is dead at point ℓ),
  - then it **definitely has that behavior** in all computations.
- However the analysis may be imprecise in this sense:

  it can answer "don't know" even when the property is true.

WHAT KIND OF REASONING CAN BE USED TO DISCOVER PROGRAM PROPERTIES?

They can involve

- **Control flow**, e.g., that point ℓ is unreachable
- **Data flow**, e.g., that the value of variable $\mathtt{x}$ at point ℓ cannot affect the program's final output.

A useful classification: dimension 1 = past/future, dimension 2 = may/must.

- **computational pasts**, e.g., that $\mathtt{x}$ equals $\mathtt{a}$ if control point ℓ is reached
- **computational futures**, e.g., that variable $\mathtt{x}$ is dead at control point ℓ
- **all-path, or "must" properties**, e.g., a past all-path property:

  "variable $\mathtt{x}$ is initialised"

  i.e., $\mathtt{x}$ was set on every computation path from start to current point ℓ
- **some-path, or "may" properties**, e.g., a future some-path property:
  "variable $\mathtt{x}$ is live", i.e., there exists a computation path from current point ℓ to the program end

OVERVIEW

A program analysis will compute a "program-point-centric" analysis that binds information to each program point ℓ.

The program properties at a program point ℓ are

- determined by
  - the computational future

    (of computations that get as far as ℓ); or
  - the computational past
- determined by the set of
  - all computation paths from (or to) ℓ, or by
  - the existence of at least one computation path from (or to) ℓ

OVERVIEW

A program analysis will compute a “program-point-centric” analysis that binds information to each program point ℓ.

Such information (almost always in the literature)

- is finitely (and feasibly!) computable
- is computed uniformly, i.e., for all the source program’s program points.
- Adjacent program points will have properties that are related, e.g., by classic flow equations of dataflow analysis for compiler construction.

An analogy: heat flow equations.

(though heat flows 2-ways, while program flows are asymmetric.)

SOME NOTATIONS USED IN THE BOOK

$\ell \in \mathrm{Lab}$	**the set of all labels**
$x, y, z \in \mathrm{Var}$	**the set of all variables**
$S \in \mathrm{Stmt}$	**the set of all statements**
$a \in \mathrm{AExp}$	**the set of all arithmetic expressions**
$b \in \mathrm{BExp}$	**the set of all Boolean expressions**
$e \in \mathrm{Exp}$	**the set of all expressions (arithmetic or Boolean)**

ABSTRACT SYNTAX

$$
\begin{array}{lcl}
a & ::= & x \mid n \mid a_1\ op_1\ a_2 \\
b & ::= & \texttt{true} \mid \texttt{false} \mid \texttt{not}\ b \mid b_1\ op_b\ b_2 \mid a_1\ op_r\ a_2 \\
S & ::= & [x := a]^\ell \mid [\texttt{skip}]^\ell \mid S_1\ ;S_2 \\
 & \mid & \texttt{if}\ [b]^\ell\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \mid \texttt{while}\ [b]^\ell\ \texttt{do}\ S \\
 & & \\
B & ::= & [x := a]^\ell \mid [\texttt{skip}]^\ell \mid [b]^\ell
\end{array}
$$

For our slides: we only think of flow charts containing labeled blocks B, don't deal with statements that contain other statements. (Doesn't lose information, and saves notation!)

Generic versus concrete:

$[x := a]^\ell$	**Math font for generic program fragments, e.g., x ranges over all variables**
$[\texttt{x:=x+1}]^7$	**Teletype font for concrete program fragments, e.g., the LHS is the concrete variable "x"**

A FEW MORE NOTATIONS

$\mathbf{Lab}_*$	**the set of all labels in the program currently being analysed**
$\mathbf{Var}_*$	**the set of all variables in the program currently being analysed**
$\mathbf{Stmt}_*$	**the set of all statements in the program currently being analysed**
$\mathbf{AExp}_*$	**the set of all arithmetic expressions in the program currently being a**
$\mathbf{BExp}_*$	**the set of all Boolean expressions in the program currently being ana**

4 USEFUL EXAMPLES OF DATA FLOW ANALYSIS

Type of flow equations:	What's analysed	
$F : \text{Lab}_* \longrightarrow$ dataflow lattice L	Time dependency	Path modality
$RD : \text{Lab}_* \longrightarrow \mathcal{P}(\text{Var}_* \times \text{Lab}_*^?)$	past	$\exists$
$LV : \text{Lab}_* \longrightarrow \mathcal{P}(\text{Var}_*)$	future	$\exists$
$AE : \text{Lab}_* \longrightarrow \mathcal{P}(\text{Exp}_*)$	past	$\forall$
$VB : \text{Lab}_* \longrightarrow \mathcal{P}(\text{Exp}_*)$	future	$\forall$

RD = Reaching definitions (used for constant propagation)

LV = Live variables (used for dead code elimination)

AE = Available expressions (to avoid recomputing expressions)

VB = Very busy expressions (save expression values for later use)

INTUITIVE EXPLANATION: LIVE VARIABLES

Type of flow equations:	What's analysed		How it's computed	
$F : \mathrm{Lab}_* \longrightarrow$ dataflow lattice L	Time dependency	Path modality	Data flow	Kind of fixpoint
$LV : \mathrm{Lab}_* \longrightarrow \mathcal{P}(\mathrm{Var}_*)$	future	$\exists$	backward	least

Variable x is live at program point ℓ if there exists a flow chart path from ℓ to some usage of variable x. Things to notice:

- it's about what can happen in the future
- along at least one path ($\exists$)

Optimisation enabled by live variable analysis:

> If x is not live at point ℓ, then the register / memory cell containg the value of x may be used for another value

Net effect: to reduce memory or register usage.

INTUITIVE EXPLANATION: AVAILABLE EXPRESSIONS

Type of flow equations:	What's analysed		How it's computed	
$F : \mathrm{Lab}_* \longrightarrow$ dataflow lattice L	Time dependency	Path modality	Data flow	Kind of fixpoint
$AE : \mathrm{Lab}_* \rightarrow \mathcal{P}(\mathrm{Exp}_*)$	past	$\forall$	forward	greatest

Expression e is available at program point ℓ if on all flow chart paths to ℓ the value of e has been computed, and no variable in e has been changed.

Things to notice:

- it's about what did happen in the past
- and along all paths to ℓ ($\forall$)

Optimisation enabled by live variable analysis:

If e is available at point ℓ, then (generate code to) fetch the value that has already been computed.

Net effect: generate smaller code.

INTUITIVE EXPLANATION: VERY BUSY EXPRESSIONS

Type of flow equations:	What's analysed		How it's computed	
$F : \mathrm{Lab}_* \longrightarrow$ dataflow lattice L	Time dependency	Path modality	Data flow	Kind of fixpoint
$VB \; : \mathrm{Lab}_* \longrightarrow \mathcal{P}(\mathrm{Exp}_*)$	future	$\forall$	backward	greatest

Expression e is very busy at program point ℓ if the value of e will be used on all flow chart paths from ℓ. Things to notice:

- it's about what will happen in the future
- and along all paths from ℓ ($\forall$)

Optimisation enabled by very busy expression analysis:

It can pay to keep the value of e in a register instead of memory.

Net effect: generate faster code.

INTUITIVE EXPLANATION: REACHING DEFINITIONS

Type of flow equations:	What's analysed		How it's computed	
$F : \mathrm{Lab}_* \to$ dataflow lattice L	Time dependency	Path modality	Data flow	Kind of fixpoint
$RD : \mathrm{Lab}_* \to \mathcal{P}(\mathrm{Var}_* \times \mathrm{Lab}_*^?)$	past	$\exists$	forward	least

A pair (x, ℓ_0) can reach program point ℓ if

- there is a statement $[x := e]^{\ell_0}$, and
- there is a path from ℓ_0 to ℓ, and
- variable x is not changed on the path

Things to notice:

- it's about what happened in the past along at least one path to ℓ ($\exists$)

Optimisation enabled by reaching definition analysis: constant propagation

Net effect: generate faster code.

SEMANTIC FOUNDATION

- **State**: a state is a function $\sigma : \mathrm{Var} \rightarrow \mathrm{Z}$. Also known as a **store**.
  Idea: the current value of variable x is $\sigma(x)$.
- A computational **configuration** is a pair $\langle S, \sigma \rangle$ where S is a statement (what is remaining to execute) and σ is the current state.
- A one-step **transition** has form

  $\langle S, \sigma \rangle \rightarrow \langle S', \sigma' \rangle$ or, if program stops: $\langle S, \sigma \rangle \rightarrow \sigma'$

  Details omitted today, but what you would expect. Here there is a **data flow** from σ to σ'
- Each program defines a set of **computations**. A computation is either
  - **a terminating computation**: a finite sequence

    $$\langle S_1, \sigma_1 \rangle \rightarrow \langle S_2, \sigma_2 \rangle \rightarrow \ldots \langle S_n, \sigma_n \rangle \rightarrow \sigma_{n+1}$$

    or
  - **a looping computation**: an infinite sequence

    $$\langle S_1, \sigma_1 \rangle \rightarrow \langle S_2, \sigma_2 \rangle \rightarrow \ldots$$

THE MAIN PROBLEM OF DFA

Given a program, to find a description of the data flow at each label ℓ. In this book, for analysis A:

- $A_{entry}(\ell)$ = flow information at the entry to statement $[B]^{\ell}$
- $A_{exit}(\ell)$ = flow information at the exit from statement $[B]^{\ell}$

Suppose program has the form:

$$[B_1]^{\ell_1}\ [B_2]^{\ell_2}\ \ldots\ [B_n]^{\ell_n}$$

Then a program description will have the form:

$$A_{entry} : \mathrm{Lab}_* \longrightarrow L \text{ and } A_{exit} : \mathrm{Lab}_* \longrightarrow L$$

where L is a complete lattice. Different lattices for different flow properties.

Flow lattice: a structure $L = (L, \sqsubseteq, \sqcup, \sqcap, \bot, \top))$.

A PAST ANALYSIS: REACHING DEFINITIONS FOR X!

Program:

`[y:=x]`^{1}`;[z:=1]`^{2}`; while [y>1]`^{3}` do ([z:=z*y]`^{4}`;[y:=y-1]`^{5}`); [y:=0]`^{6}`;`

Reaching definitions lattice:

$$L = (\ \mathcal{P}(\{\mathrm{x,y,z}\} \times \{1,2,3,4,5,6,?\})\ , \sqsubseteq, \sqcup, \sqcap, \bot, \top)$$

$(x, \ell_0) \in RD\_(\ell)$ if for some computation path from ℓ_0 to ℓ

- x was assigned at point ℓ_0, and (jargon: "defined")
- x was not re-assigned before point ℓ (i.e., the assignment "reaches" ℓ)

Uninitalised variables: are "reached" from point "?"

ℓ	$RD_{entry}(\ell)$	$RD_{exit}(\ell)$
1	$\{(\mathrm{x},?),(\mathrm{y},?),(\mathrm{z},?)\}$	$\{(\mathrm{x},?),(\mathrm{y},1),(\mathrm{z},?)\}$
2	$\{(\mathrm{x},?),(\mathrm{y},1),(\mathrm{z},?)\}$	$\{(\mathrm{x},?),(\mathrm{y},1),(\mathrm{z},2)\}$
3	$\{(\mathrm{x},?),(\mathrm{y},1),(\mathrm{y},5),(\mathrm{z},2),(\mathrm{z},4)\}$	$\{(\mathrm{x},?),(\mathrm{y},1),(\mathrm{y},5),(\mathrm{z},2),(\mathrm{z},4)\}$
4	$\{(\mathrm{x},?),(\mathrm{y},1),(\mathrm{y},5),(\mathrm{z},2),(\mathrm{z},4)\}$	$\{(\mathrm{x},?),(\mathrm{y},1),(\mathrm{y},5),(\mathrm{z},4)\}$
5	$\{(\mathrm{x},?),(\mathrm{y},1),(\mathrm{y},5),(\mathrm{z},4)\}$	$\{(\mathrm{x},?),(\mathrm{y},5),(\mathrm{z},4)\}$
6	$\{(\mathrm{x},?),(\mathrm{y},1),(\mathrm{y},5),(\mathrm{z},2),(\mathrm{z},4)\}$	$\{(\mathrm{x},?),(\mathrm{y},6),(\mathrm{z},2),(\mathrm{z},4)\}$

A FUTURE ANALYSIS: LIVE VARIABLES
(for the same program to compute x!)

Program:

[y:=x]^1;[z:=1]^2; while [y>1]^3 do ([z:=z*y]^4;[y:=y-1]^5); [y:=0]^6;

Live variable lattice:

$$L = (\ \mathcal{P}(\{\mathtt{x},\mathtt{y},\mathtt{z}\})\ , \sqsubseteq, \sqcup, \sqcap, \bot, \top)$$

Variable x is live if $\exists$ computation path with a future reference to x.

Assume: no variables are live at program exit.

ℓ	$LV_{entry}(\ell)$	$LV_{exit}(\ell)$
1	$\{\mathtt{x}\}$	$\{\mathtt{y}\}$
2	$\{\mathtt{y}\}$	$\{\mathtt{y},\mathtt{z}\}$
3	$\{\mathtt{y},\mathtt{z}\}$	$\{\mathtt{y},\mathtt{z}\}$
4	$\{\mathtt{y},\mathtt{z}\}$	$\{\mathtt{y},\mathtt{z}\}$
5	$\{\mathtt{y},\mathtt{z}\}$	$\{\mathtt{y},\mathtt{z}\}$
6	$\emptyset$	$\emptyset$

LIVE VARIABLE FLOW EQUATIONS:

(for the same program to compute x!)

```
[y:=x]^1;[z:=1]^2; while [y>1]^3 do ([z:=z*y]^4;[y:=y-1]^5);[y:=0]^6;
```

$$
\begin{aligned}
LV_{entry}(1) &= LV_{exit}(1) \setminus \{\mathtt{y}\} \cup \{\mathtt{x}\} \\
LV_{entry}(2) &= LV_{exit}(2) \setminus \{\mathtt{z}\} \\
LV_{entry}(3) &= LV_{exit}(3) \cup \{\mathtt{y}\} \\
LV_{entry}(4) &= LV_{exit}(4) \cup \{\mathtt{y}, \mathtt{z}\} \\
LV_{entry}(5) &= LV_{exit}(5) \cup \{\mathtt{y}\} \\
LV_{entry}(6) &= LV_{exit}(6) \setminus \{\mathtt{y}\} \\
LV_{exit}(1) &= LV_{entry}(2) \\
LV_{exit}(2) &= LV_{entry}(3) \\
LV_{exit}(3) &= LV_{entry}(4) \cup LV_{entry}(6) \\
LV_{exit}(4) &= LV_{entry}(5) \\
LV_{exit}(5) &= LV_{entry}(3) \\
LV_{exit}(6) &= \emptyset
\end{aligned}
$$

WHAT ON EARTH IS GOING ON?

- What is being defined by these equations ?
- What data flow logic is being expressed?
- How can the equations be solved ?

The equations define the values of in all 12 program point descriptions

$$LV_{entry}(1), \ldots, LV_{entry}(6), LV_{exit}(1), \ldots, LV_{exit}(6)$$

in terms of each other.

This is a recursive system of data flow equations to describe the program's computational behavior.

Solution to the equation system: This is called a fixpoint.

Type of a solution to the equation system: L^{12}, where L is the description data flow lattice.

Type of the equation system itself:

$$F : L^{12} \longrightarrow L^{12}$$

FLOW EQUATION DIMENSIONS

- **Time dependence**. Possibilities:
  - Future analysis: the property depends on the computational future.
    Computed by backward data flow.
  - Past analysis: the property depends on the computational past.
    Computed by forward data flow. – "must" or "may" dependence:
- **Path modality dependence**. Possibilities:
  - may path dependence (for some path)
  - must path dependence (for all paths)
- These make four combinations. For example:
  - Both LV and RD are may path dependencies
  - Live variables LV is a future analysis (= backward data flow)
  - Reaching definitions RD is a past analysis (= forward data flow)

FLOW EQUATIONS: REFLECT THE 4 COMBINATIONS

Future/past: what is defined in terms of what in the equations, e.g.,

future: $LV_{entry}(\ell) = \ldots LV_{exit}(\ell) \ldots$
past: $LV_{exit}(\ell) = \ldots LV_{entry}(\ell) \ldots$

All-paths/some-path: find greatest or least fixpoint solution to equations

Fixpoints: *lfp* (least fixpoint) for $\exists$ path dependence

$$lfp(F) = \bigsqcup_{n \to \infty} F^n(\bot, \bot, \ldots, \bot)$$

gfp (greatest fixpoint) for $\forall$ path dependence

$$gfp(F) = \sqcap_{n \to \infty} F^n(\top, \top, \ldots, \top)$$

Combining flows from several blocks into one:

- Use $\sqcup$ when computing least fixpoint (some-path properties)
- Use $\sqcap$ when computing greatest fixpoint (all-path properties)

4 EXAMPLES OF THE 4 COMBINATIONS

Type of flow equations:	What's analysed		How it's computed	
$F : \text{Lab}_* \to$ dataflow lattice L	Time dependency	Path modality	Data flow	Kind of fixpoint
$RD\ : \text{Lab}_* \to \mathcal{P}(\text{Var}_* \times \text{Lab}_*^?)$	past	$\exists$	forward	least
$LV\ : \text{Lab}_* \to \mathcal{P}(\text{Var}_*)$	future	$\exists$	backward	least
$AE\ : \text{Lab}_* \to \mathcal{P}(\text{Exp}_*)$	past	$\forall$	forward	greatest
$VB\ : \text{Lab}_* \to \mathcal{P}(\text{Exp}_*)$	future	$\forall$	backward	greatest

RD = Reaching definitions
LV = Live variables
AE = Available expressions
VB = Very busy expressions

RELATIONS TO LATTICES ETC. FROM APPENDIX A

Form of the data flow equation system:

$$(X_1, X_2, \ldots, X_{2n}) = (e_1(\vec{X}), e_2(\vec{X}), \ldots, e_{2n}(\vec{X}))$$

where set expressions $e_e, \ldots, e_{2n}$ are built from $X_1, X_2, \ldots, X_{2n}$ by set operations such as $\cup, \cap, \setminus$ and constants.

This defines a function

$$F : \mathcal{P}(D)^{2n} \to \mathcal{P}(D)^{2n}$$

(where D = set of descriptions, n = number of labels)

on the lattice

$$L = (L, \sqsubseteq, \sqcup, \sqcap, \bot, \top)) = (\mathcal{P}(D), \subseteq, \cup, \cap, \emptyset, D))$$

Fixpoints: $lfp(F) = \bigsqcup_{n\to\infty} F^n(\bot, \bot, \ldots, \bot)$, $gfp(F) = \sqcap_{n\to\infty} F^n(\top, \ldots, \top)$

1. $\mathcal{P}(D)$ is a lattice, so $F(X_1, X_2, \ldots, X_{2n})$ exists.
2. $\mathcal{P}(D)$ is complete, so $lfp(F), gfp(F)$ both exist.
3. Ascending (descending) chain condition: ensures that $lfp(F), gfp(F)$ are finitely computable.

CHAOTIC ITERATION

Effect is to compute the least (or greatest) fixpoint by repeatedly applying the equations

- **Apply them in any order**
- **until no sets can be changed**
- **Initialisation of the sets:**
  - **Least fixpoint: start with every set empty ($\perp$ of the lattice)**
  - **Greatest fixpoint: start with every set equal to ($\top$ of the lattice)**
- **Amazing fact: it doesn't matter what order is chosen (hence the name "chaotic")**

HAND SOLUTION: LIVE VARIABLE ANALYSIS OF THE FACTORIAL PROGRAM BY CHAOTIC ITERATION 1

[y:=x]^1;[z:=1]^2; while [y>1]^3 do ([z:=z*y]^4;[y:=y-1]^5);[y:=0]^6;

$$
\begin{aligned}
LV_{entry}(1) &= LV_{exit}(1) \setminus \{\mathtt{y}\} \cup \{\mathtt{x}\} \\
LV_{entry}(2) &= LV_{exit}(2) \setminus \{\mathtt{z}\} \\
LV_{entry}(3) &= LV_{exit}(3) \cup \{\mathtt{y}\} \\
LV_{entry}(4) &= LV_{exit}(4) \cup \{\mathtt{y}, \mathtt{z}\} \\
LV_{entry}(5) &= LV_{exit}(5) \cup \{\mathtt{y}\} \\
LV_{entry}(6) &= LV_{exit}(6) \setminus \{\mathtt{y}\} \\
LV_{exit}(1) &= LV_{entry}(2) \\
LV_{exit}(2) &= LV_{entry}(3) \\
LV_{exit}(3) &= LV_{entry}(4) \cup LV_{entry}(6) \\
LV_{exit}(4) &= LV_{entry}(5) \\
LV_{exit}(5) &= LV_{entry}(3) \\
LV_{exit}(6) &= \emptyset
\end{aligned}
$$

ℓ	$LV_{entry}(\ell)$	$LV_{exit}(\ell)$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\emptyset$
3	$\emptyset$	$\emptyset$
4	$\emptyset$	$\emptyset$
5	$\emptyset$	$\emptyset$
6	$\emptyset$	$\emptyset$

HAND SOLUTION: LIVE VARIABLE ANALYSIS OF THE FACTORIAL PROGRAM BY CHAOTIC ITERATION 2

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

$$
\begin{aligned}
LV_{entry}(1) &= LV_{exit}(1) \setminus \{\mathtt{y}\} \cup \{\mathtt{x}\}\\
LV_{entry}(2) &= LV_{exit}(2) \setminus \{\mathtt{z}\}\\
LV_{entry}(3) &= LV_{exit}(3) \cup \{\mathtt{y}\}\\
LV_{entry}(4) &= LV_{exit}(4) \cup \{\mathtt{y},\mathtt{z}\}\\
LV_{entry}(5) &= LV_{exit}(5) \cup \{\mathtt{y}\}\\
LV_{entry}(6) &= LV_{exit}(6) \setminus \{\mathtt{y}\}\\
LV_{exit}(1) &= LV_{entry}(2)\\
LV_{exit}(2) &= LV_{entry}(3)\\
LV_{exit}(3) &= LV_{entry}(4) \cup LV_{entry}(6)\\
LV_{exit}(4) &= LV_{entry}(5)\\
LV_{exit}(5) &= LV_{entry}(3)\\
LV_{exit}(6) &= \emptyset
\end{aligned}
$$

ℓ	$LV_{entry}(\ell)$	$LV_{exit}(\ell)$
1	$\{\mathtt{x}\}$	$\emptyset$
2	$\emptyset$	$\emptyset$
3	$\{\mathtt{y}\}$	$\emptyset$
4	$\{\mathtt{y},\mathtt{z}\}$	$\emptyset$
5	$\{\mathtt{y}\}$	$\emptyset$
6	$\emptyset$	$\emptyset$

HAND SOLUTION: LIVE VARIABLE ANALYSIS OF THE FACTORIAL PROGRAM BY CHAOTIC ITERATION 3

```
[y:=x]^1;[z:=1]^2; while [y>1]^3 do ([z:=z*y]^4;[y:=y-1]^5);[y:=0]^6;
```

$$
\begin{aligned}
LV_{entry}(1) &= LV_{exit}(1) \setminus \{\mathtt{y}\} \cup \{\mathtt{x}\} \\
LV_{entry}(2) &= LV_{exit}(2) \setminus \{\mathtt{z}\} \\
LV_{entry}(3) &= LV_{exit}(3) \cup \{\mathtt{y}\} \\
LV_{entry}(4) &= LV_{exit}(4) \cup \{\mathtt{y},\mathtt{z}\} \\
LV_{entry}(5) &= LV_{exit}(5) \cup \{\mathtt{y}\} \\
LV_{entry}(6) &= LV_{exit}(6) \setminus \{\mathtt{y}\} \\
LV_{exit}(1) &= LV_{entry}(2) \\
LV_{exit}(2) &= LV_{entry}(3) \\
LV_{exit}(3) &= LV_{entry}(4) \cup LV_{entry}(6) \\
LV_{exit}(4) &= LV_{entry}(5) \\
LV_{exit}(5) &= LV_{entry}(3) \\
LV_{exit}(6) &= \emptyset
\end{aligned}
$$

ℓ	$LV_{entry}(\ell)$	$LV_{exit}(\ell)$
1	$\{\mathtt{x}\}$	$\emptyset$
2	$\emptyset$	$\{\mathtt{y}\}$
3	$\{\mathtt{y}\}$	$\emptyset$
4	$\{\mathtt{y},\mathtt{z}\}$	$\emptyset$
5	$\{\mathtt{y}\}$	$\emptyset$
6	$\emptyset$	$\emptyset$

HAND SOLUTION: LIVE VARIABLE ANALYSIS OF THE FACTORIAL PROGRAM BY CHAOTIC ITERATION 4

`[y:=x]`^{1}`;[z:=1]`^{2}`; while [y>1]`^{3}` do ([z:=z*y]`^{4}`;[y:=y-1]`^{5}`);[y:=0]`^{6}`;`

$$
\begin{aligned}
LV_{entry}(1) &= LV_{exit}(1) \setminus \{\mathtt{y}\} \cup \{\mathtt{x}\} \\
LV_{entry}(2) &= LV_{exit}(2) \setminus \{\mathtt{z}\} \\
LV_{entry}(3) &= LV_{exit}(3) \cup \{\mathtt{y}\} \\
LV_{entry}(4) &= LV_{exit}(4) \cup \{\mathtt{y},\mathtt{z}\} \\
LV_{entry}(5) &= LV_{exit}(5) \cup \{\mathtt{y}\} \\
LV_{entry}(6) &= LV_{exit}(6) \setminus \{\mathtt{y}\} \\
LV_{exit}(1) &= LV_{entry}(2) \\
LV_{exit}(2) &= LV_{entry}(3) \\
LV_{exit}(3) &= LV_{entry}(4) \cup LV_{entry}(6) \\
LV_{exit}(4) &= LV_{entry}(5) \\
LV_{exit}(5) &= LV_{entry}(3) \\
LV_{exit}(6) &= \emptyset
\end{aligned}
$$

ℓ	$LV_{entry}(\ell)$	$LV_{exit}(\ell)$
1	$\{\mathtt{x}\}$	$\emptyset$
2	$\emptyset$	$\{\mathtt{y}\}$
3	$\{\mathtt{y}\}$	$\{\mathtt{y},\mathtt{z}\}$
4	$\{\mathtt{y},\mathtt{z}\}$	$\emptyset$
5	$\{\mathtt{y}\}$	$\emptyset$
6	$\emptyset$	$\emptyset$

HAND SOLUTION: LIVE VARIABLE ANALYSIS OF THE FACTORIAL PROGRAM BY CHAOTIC ITERATION 5

`[y:=x]`^{1}`;[z:=1]`^{2}`; while [y>1]`^{3}` do ([z:=z*y]`^{4}`;[y:=y-1]`^{5}`);[y:=0]`^{6}`;`

$$
\begin{aligned}
LV_{entry}(1) &= LV_{exit}(1) \setminus \{\mathtt{y}\} \cup \{\mathtt{x}\} \\
LV_{entry}(2) &= LV_{exit}(2) \setminus \{\mathtt{z}\} \\
LV_{entry}(3) &= LV_{exit}(3) \cup \{\mathtt{y}\} \\
LV_{entry}(4) &= LV_{exit}(4) \cup \{\mathtt{y}, \mathtt{z}\} \\
LV_{entry}(5) &= LV_{exit}(5) \cup \{\mathtt{y}\} \\
LV_{entry}(6) &= LV_{exit}(6) \setminus \{\mathtt{y}\} \\
LV_{exit}(1) &= LV_{entry}(2) \\
LV_{exit}(2) &= LV_{entry}(3) \\
LV_{exit}(3) &= LV_{entry}(4) \cup LV_{entry}(6) \\
LV_{exit}(4) &= LV_{entry}(5) \\
LV_{exit}(5) &= LV_{entry}(3) \\
LV_{exit}(6) &= \emptyset
\end{aligned}
$$

ℓ	$LV_{entry}(\ell)$	$LV_{exit}(\ell)$
1	$\{\mathtt{x}\}$	$\emptyset$
2	$\emptyset$	$\{\mathtt{y}\}$
3	$\{\mathtt{y}\}$	$\{\mathtt{y}, \mathtt{z}\}$
4	$\{\mathtt{y}, \mathtt{z}\}$	$\{\mathtt{y}\}$
5	$\{\mathtt{y}\}$	$\emptyset$
6	$\emptyset$	$\emptyset$

HAND SOLUTION: LIVE VARIABLE ANALYSIS OF THE FACTORIAL PROGRAM BY CHAOTIC ITERATION 6

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

$$
\begin{aligned}
LV_{entry}(1) &= LV_{exit}(1) \setminus \{\mathtt{y}\} \cup \{\mathtt{x}\} \\
LV_{entry}(2) &= LV_{exit}(2) \setminus \{\mathtt{z}\} \\
LV_{entry}(3) &= LV_{exit}(3) \cup \{\mathtt{y}\} \\
LV_{entry}(4) &= LV_{exit}(4) \cup \{\mathtt{y},\mathtt{z}\} \\
LV_{entry}(5) &= LV_{exit}(5) \cup \{\mathtt{y}\} \\
LV_{entry}(6) &= LV_{exit}(6) \setminus \{\mathtt{y}\} \\
LV_{exit}(1) &= LV_{entry}(2) \\
LV_{exit}(2) &= LV_{entry}(3) \\
LV_{exit}(3) &= LV_{entry}(4) \cup LV_{entry}(6) \\
LV_{exit}(4) &= LV_{entry}(5) \\
LV_{exit}(5) &= LV_{entry}(3) \\
LV_{exit}(6) &= \emptyset
\end{aligned}
$$

ℓ	$LV_{entry}(\ell)$	$LV_{exit}(\ell)$
1	$\{\mathtt{x}\}$	$\emptyset$
2	$\emptyset$	$\{\mathtt{y}\}$
3	$\{\mathtt{y}\}$	$\{\mathtt{y},\mathtt{z}\}$
4	$\{\mathtt{y},\mathtt{z}\}$	$\{\mathtt{y}\}$
5	$\{\mathtt{y}\}$	$\{\mathtt{y}\}$
6	$\emptyset$	$\emptyset$

HAND SOLUTION: LIVE VARIABLE ANALYSIS OF THE FACTORIAL PROGRAM BY CHAOTIC ITERATION 7

`[y:=x]`^{1}`;[z:=1]`^{2}`; while [y>1]`^{3}` do ([z:=z*y]`^{4}`;[y:=y-1]`^{5}`);[y:=0]`^{6}`;`

$$
\begin{aligned}
LV_{entry}(1) &= LV_{exit}(1) \setminus \{\mathtt{y}\} \cup \{\mathtt{x}\} \\
LV_{entry}(2) &= LV_{exit}(2) \setminus \{\mathtt{z}\} \\
LV_{entry}(3) &= LV_{exit}(3) \cup \{\mathtt{y}\} \\
LV_{entry}(4) &= LV_{exit}(4) \cup \{\mathtt{y}, \mathtt{z}\} \\
LV_{entry}(5) &= LV_{exit}(5) \cup \{\mathtt{y}\} \\
LV_{entry}(6) &= LV_{exit}(6) \setminus \{\mathtt{y}\} \\
LV_{exit}(1) &= LV_{entry}(2) \\
LV_{exit}(2) &= LV_{entry}(3) \\
LV_{exit}(3) &= LV_{entry}(4) \cup LV_{entry}(6) \\
LV_{exit}(4) &= LV_{entry}(5) \\
LV_{exit}(5) &= LV_{entry}(3) \\
LV_{exit}(6) &= \emptyset
\end{aligned}
$$

ℓ	$LV_{entry}(\ell)$	$LV_{exit}(\ell)$
1	$\{\mathtt{x}\}$	$\emptyset$
2	$\{\mathtt{y}\}$	$\{\mathtt{y}\}$
3	$\{\mathtt{y}\}$	$\{\mathtt{y}, \mathtt{z}\}$
4	$\{\mathtt{y}, \mathtt{z}\}$	$\{\mathtt{y}\}$
5	$\{\mathtt{y}\}$	$\{\mathtt{y}\}$
6	$\emptyset$	$\emptyset$

HAND SOLUTION: LIVE VARIABLE ANALYSIS OF THE FACTORIAL PROGRAM BY CHAOTIC ITERATION 8

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

$$
\begin{aligned}
LV_{entry}(1) &= LV_{exit}(1) \setminus \{\texttt{y}\} \cup \{\texttt{x}\} \\
LV_{entry}(2) &= LV_{exit}(2) \setminus \{\texttt{z}\} \\
LV_{entry}(3) &= LV_{exit}(3) \cup \{\texttt{y}\} \\
LV_{entry}(4) &= LV_{exit}(4) \cup \{\texttt{y}, \texttt{z}\} \\
LV_{entry}(5) &= LV_{exit}(5) \cup \{\texttt{y}\} \\
LV_{entry}(6) &= LV_{exit}(6) \setminus \{\texttt{y}\} \\
LV_{exit}(1) &= LV_{entry}(2) \\
LV_{exit}(2) &= LV_{entry}(3) \\
LV_{exit}(3) &= LV_{entry}(4) \cup LV_{entry}(6) \\
LV_{exit}(4) &= LV_{entry}(5) \\
LV_{exit}(5) &= LV_{entry}(3) \\
LV_{exit}(6) &= \emptyset
\end{aligned}
$$

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

HAND SOLUTION: LIVE VARIABLE ANALYSIS OF THE FACTORIAL PROGRAM BY CHAOTIC ITERATION 9

[y:=x]^1;[z:=1]^2; while [y>1]^3 do ([z:=z*y]^4;[y:=y-1]^5);[y:=0]^6;

$$
\begin{aligned}
LV_{entry}(1) &= LV_{exit}(1) \setminus \{\mathtt{y}\} \cup \{\mathtt{x}\} \\
LV_{entry}(2) &= LV_{exit}(2) \setminus \{\mathtt{z}\} \\
LV_{entry}(3) &= LV_{exit}(3) \cup \{\mathtt{y}\} \\
LV_{entry}(4) &= LV_{exit}(4) \cup \{\mathtt{y},\mathtt{z}\} \\
LV_{entry}(5) &= LV_{exit}(5) \cup \{\mathtt{y}\} \\
LV_{entry}(6) &= LV_{exit}(6) \setminus \{\mathtt{y}\} \\
LV_{exit}(1) &= LV_{entry}(2) \\
LV_{exit}(2) &= LV_{entry}(3) \\
LV_{exit}(3) &= LV_{entry}(4) \cup LV_{entry}(6) \\
LV_{exit}(4) &= LV_{entry}(5) \\
LV_{exit}(5) &= LV_{entry}(3) \\
LV_{exit}(6) &= \emptyset
\end{aligned}
$$

ℓ	$LV_{entry}(\ell)$	$LV_{exit}(\ell)$
1	$\{\mathtt{x}\}$	$\{\mathtt{y}\}$
2	$\{\mathtt{y}\}$	$\{\mathtt{y}\}$
3	$\{\mathtt{y},\mathtt{z}\}$	$\{\mathtt{y},\mathtt{z}\}$
4	$\{\mathtt{y},\mathtt{z}\}$	$\{\mathtt{y}\}$
5	$\{\mathtt{y}\}$	$\{\mathtt{y}\}$
6	$\emptyset$	$\emptyset$

HAND SOLUTION: LIVE VARIABLE ANALYSIS OF THE FACTORIAL PROGRAM BY CHAOTIC ITERATION 10

[y:=x]^1;[z:=1]^2; while [y>1]^3 do ([z:=z*y]^4;[y:=y-1]^5);[y:=0]^6;

$$
\begin{aligned}
LV_{entry}(1) &= LV_{exit}(1) \setminus \{\mathtt{y}\} \cup \{\mathtt{x}\}\\
LV_{entry}(2) &= LV_{exit}(2) \setminus \{\mathtt{z}\}\\
LV_{entry}(3) &= LV_{exit}(3) \cup \{\mathtt{y}\}\\
LV_{entry}(4) &= LV_{exit}(4) \cup \{\mathtt{y}, \mathtt{z}\}\\
LV_{entry}(5) &= LV_{exit}(5) \cup \{\mathtt{y}\}\\
LV_{entry}(6) &= LV_{exit}(6) \setminus \{\mathtt{y}\}\\
LV_{exit}(1) &= LV_{entry}(2)\\
LV_{exit}(2) &= LV_{entry}(3)\\
LV_{exit}(3) &= LV_{entry}(4) \cup LV_{entry}(6)\\
LV_{exit}(4) &= LV_{entry}(5)\\
LV_{exit}(5) &= LV_{entry}(3)\\
LV_{exit}(6) &= \emptyset
\end{aligned}
$$

ℓ	$LV_{entry}(\ell)$	$LV_{exit}(\ell)$
1	{x}	{y}
2	{y}	{y, z}
3	{y, z}	{y, z}
4	{y, z}	{y}
5	{y}	{y}
6	$\emptyset$	$\emptyset$

HAND SOLUTION: LIVE VARIABLE ANALYSIS OF THE FACTORIAL PROGRAM BY CHAOTIC ITERATION 11

[y:=x]^1;[z:=1]^2; while [y>1]^3 do ([z:=z*y]^4;[y:=y-1]^5);[y:=0]^6;

$$
\begin{aligned}
LV_{entry}(1) &= LV_{exit}(1) \setminus \{\mathrm{y}\} \cup \{\mathrm{x}\} \\
LV_{entry}(2) &= LV_{exit}(2) \setminus \{\mathrm{z}\} \\
LV_{entry}(3) &= LV_{exit}(3) \cup \{\mathrm{y}\} \\
LV_{entry}(4) &= LV_{exit}(4) \cup \{\mathrm{y}, \mathrm{z}\} \\
LV_{entry}(5) &= LV_{exit}(5) \cup \{\mathrm{y}\} \\
LV_{entry}(6) &= LV_{exit}(6) \setminus \{\mathrm{y}\} \\
LV_{exit}(1) &= LV_{entry}(2) \\
LV_{exit}(2) &= LV_{entry}(3) \\
LV_{exit}(3) &= LV_{entry}(4) \cup LV_{entry}(6) \\
LV_{exit}(4) &= LV_{entry}(5) \\
LV_{exit}(5) &= LV_{entry}(3) \\
LV_{exit}(6) &= \emptyset
\end{aligned}
$$

ℓ	$LV_{entry}(\ell)$	$LV_{exit}(\ell)$
1	$\{\mathrm{x}\}$	$\{\mathrm{y}\}$
2	$\{\mathrm{y}\}$	$\{\mathrm{y}, \mathrm{z}\}$
3	$\{\mathrm{y}, \mathrm{z}\}$	$\{\mathrm{y}, \mathrm{z}\}$
4	$\{\mathrm{y}, \mathrm{z}\}$	$\{\mathrm{y}\}$
5	$\{\mathrm{y}\}$	$\{\mathrm{y}, \mathrm{z}\}$
6	$\emptyset$	$\emptyset$

HAND SOLUTION: LIVE VARIABLE ANALYSIS OF THE FACTORIAL PROGRAM BY CHAOTIC ITERATION 12

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

$$
\begin{aligned}
LV_{entry}(1) &= LV_{exit}(1) \setminus \{\mathrm{y}\} \cup \{\mathrm{x}\} \\
LV_{entry}(2) &= LV_{exit}(2) \setminus \{\mathrm{z}\} \\
LV_{entry}(3) &= LV_{exit}(3) \cup \{\mathrm{y}\} \\
LV_{entry}(4) &= LV_{exit}(4) \cup \{\mathrm{y}, \mathrm{z}\} \\
LV_{entry}(5) &= LV_{exit}(5) \cup \{\mathrm{y}\} \\
LV_{entry}(6) &= LV_{exit}(6) \setminus \{\mathrm{y}\} \\
LV_{exit}(1) &= LV_{entry}(2) \\
LV_{exit}(2) &= LV_{entry}(3) \\
LV_{exit}(3) &= LV_{entry}(4) \cup LV_{entry}(6) \\
LV_{exit}(4) &= LV_{entry}(5) \\
LV_{exit}(5) &= LV_{entry}(3) \\
LV_{exit}(6) &= \emptyset
\end{aligned}
$$

ℓ	$LV_{entry}(\ell)$	$LV_{exit}(\ell)$
1	$\{\mathrm{x}\}$	$\{\mathrm{y}\}$
2	$\{\mathrm{y}\}$	$\{\mathrm{y}, \mathrm{z}\}$
3	$\{\mathrm{y}, \mathrm{z}\}$	$\{\mathrm{y}, \mathrm{z}\}$
4	$\{\mathrm{y}, \mathrm{z}\}$	$\{\mathrm{y}\}$
5	$\{\mathrm{y}, \mathrm{z}\}$	$\{\mathrm{y}, \mathrm{z}\}$
6	$\emptyset$	$\emptyset$

HAND SOLUTION: LIVE VARIABLE ANALYSIS OF THE FACTORIAL PROGRAM BY CHAOTIC ITERATION 13

`[y:=x]`^{1}`;[z:=1]`^{2}`; while [y>1]`^{3}` do ([z:=z*y]`^{4}`;[y:=y-1]`^{5}`);[y:=0]`^{6}`;`

$$
\begin{aligned}
LV_{entry}(1) &= LV_{exit}(1) \setminus \{\mathrm{y}\} \cup \{\mathrm{x}\} \\
LV_{entry}(2) &= LV_{exit}(2) \setminus \{\mathrm{z}\} \\
LV_{entry}(3) &= LV_{exit}(3) \cup \{\mathrm{y}\} \\
LV_{entry}(4) &= LV_{exit}(4) \cup \{\mathrm{y}, \mathrm{z}\} \\
LV_{entry}(5) &= LV_{exit}(5) \cup \{\mathrm{y}\} \\
LV_{entry}(6) &= LV_{exit}(6) \setminus \{\mathrm{y}\} \\
LV_{exit}(1) &= LV_{entry}(2) \\
LV_{exit}(2) &= LV_{entry}(3) \\
LV_{exit}(3) &= LV_{entry}(4) \cup LV_{entry}(6) \\
LV_{exit}(4) &= LV_{entry}(5) \\
LV_{exit}(5) &= LV_{entry}(3) \\
LV_{exit}(6) &= \emptyset
\end{aligned}
$$

ℓ	$LV_{entry}(\ell)$	$LV_{exit}(\ell)$
1	$\{\mathrm{x}\}$	$\{\mathrm{y}\}$
2	$\{\mathrm{y}\}$	$\{\mathrm{y}, \mathrm{z}\}$
3	$\{\mathrm{y}, \mathrm{z}\}$	$\{\mathrm{y}, \mathrm{z}\}$
4	$\{\mathrm{y}, \mathrm{z}\}$	$\{\mathrm{y}, \mathrm{z}\}$
5	$\{\mathrm{y}, \mathrm{z}\}$	$\{\mathrm{y}, \mathrm{z}\}$
6	$\emptyset$	$\emptyset$

THE ITERATION PROCESS CONVERGED! (AT LAST)

Chaotic iteration:

- **this always works, i.e., it always converges and to the same fixpoint**
- **the final result is a safe description of the program's data flow.**
- **some iteration orders converge faster than others.**

LOOKS LIKE MAGIC!
WHERE DO THE FLOW EQUATIONS COME FROM?

Short answer: the result of much experience in writing analysis phases for real compilers. We'll see some examples.

Future/past: what is defined in terms of what in the equations, e.g.,

future: $LV_{entry}(\ell) = \ldots LV_{exit}(\ell) \ldots$
past: $LV_{exit}(\ell) = \ldots LV_{entry}(\ell) \ldots$

All-paths/some-path: find greatest or least fixpoint solution to the equations

- *lfp* (least fixpoint) for $\exists$ path dependence
- *gfp* (greatest fixpoint) for $\forall$ path dependence

Combining flows from several blocks into one:

- Use $\sqcup$ when computing least fixpoint (some-path properties)
- Use $\sqcap$ when computing greatest fixpoint (all-path properties)

SEVERAL APPROACHES TO DATA FLOW ANALYSIS

- Data flow equations over a lattice (what we just saw)
- The “kill/gen” approach to data flow equations (a traditional compiler-writer’s approach)
- Constraint based analysis
- Monotone frameworks (unified lattice-theoretic viewpoint; notationally complex)
- Type and effect systems
- Abstract interpretation

"KILL/GEN" DATA FLOW EQUATIONS

For a future analysis AN:

$$AN_{entry}(\ell) = AN_{exit}(\ell) \setminus kill_{AN}(B^\ell) \sqcup gen_{AN}(B^\ell)$$

For a past analysis AN:

$$AN_{exit}(\ell) = AN_{entry}(\ell) \setminus kill_{AN}(B^\ell) \sqcup gen_{AN}(B^\ell)$$

Idea, reasoning:

- $kill_{AN}(B^\ell)$ expresses the data flow information that is over-written by statement B^ℓ
- $gen_{AN}(B^\ell)$ expresses the new data flow information that is added by statement B^ℓ

Example for live variable analysis: statement $[\mathtt{x:=y+z}]^3$ will

- Generate {y,z}, so $gen_{LV}([\mathtt{x:=y+z}]^3) = \{\mathtt{y}, \mathtt{z}\}$
- Kill x, so $kill_{LV}([\mathtt{x:=y+z}]^3) = \{\mathtt{x}\}$

MORE CONCRETELY: LIVE VARIABLE FLOW EQUATIONS

`[y:=x]`^{1}; `[z:=1]`^{2}; `while [y>1]`^{3} `do ([z:=z*y]`^{4}; `[y:=y-1]`^{5}); `[y:=0]`^{6};

$$
\begin{aligned}
LV_{entry}(1) &= LV_{exit}(1) \setminus \{\mathtt{y}\} \cup \{\mathtt{x}\} \\
LV_{entry}(2) &= LV_{exit}(2) \setminus \{\mathtt{z}\} \\
LV_{entry}(3) &= LV_{exit}(3) \cup \{\mathtt{y}\} \\
LV_{entry}(4) &= LV_{exit}(4) \setminus \{\mathtt{z}\} \cup \{\mathtt{y},\mathtt{z}\} \\
LV_{entry}(5) &= LV_{exit}(5) \setminus \{\mathtt{y}\} \cup \{\mathtt{y}\} \\
LV_{entry}(6) &= LV_{exit}(6) \setminus \{\mathtt{y}\} \\
LV_{exit}(1) &= LV_{entry}(2) \\
LV_{exit}(2) &= LV_{entry}(3) \\
LV_{exit}(3) &= LV_{entry}(4) \cup LV_{entry}(6) \\
LV_{exit}(4) &= LV_{entry}(5) \\
LV_{exit}(5) &= LV_{entry}(3) \\
LV_{exit}(6) &= \emptyset
\end{aligned}
$$

Examples: $kill_{LV}(\texttt{[z:=z*y]}^4) = \{z\}$ and $gen_{LV}(\texttt{[z:=z*y]}^4) = \{\mathtt{y},\mathtt{z}\}$

GENERAL DATA FLOW EQUATIONS: LIVE VARIABLES

$$LV_{exit}(\ell) = \begin{cases} \emptyset & \text{if } [B]^\ell \text{ a final block} \\ \bigcup\{LV_{entry}(\ell') \mid \ell' \to \ell \text{ in flow chart}\} & \text{otherwise} \end{cases}$$

$$LV_{entry}(\ell) = (LV_{exit}(\ell) \setminus kill_{LV}(B^\ell)) \cup gen_{LV}(B^\ell)$$

where B^ℓ is a block

- A future analysis, thus data flows backwards (from LV_{exit} to LV_{entry})
- An $\exists$ path analysis, thus *lfp* and use $\bigcup$ to merge branches

Some auxiliary definitions

$$\begin{aligned} kill_{LV}([x := a]^\ell) &= \{x\} \\ kill_{LV}([\texttt{skip}]^\ell) &= \emptyset \\ kill_{LV}([b]^\ell) &= \emptyset \end{aligned}$$

$$\begin{aligned} gen_{LV}([x := a]^\ell) &= FreeVariables(a) \\ gen_{LV}([\texttt{skip}]^\ell) &= \emptyset \\ gen_{LV}([b]^\ell) &= FreeVariables(b) \end{aligned}$$

GENERAL FLOW EQUATIONS: REACHING DEFINITIONS

$$RD_{entry}(\ell) = \begin{cases} \{(x, ?) \mid x \in \mathit{FreeVariables}(S)\} & \text{if } [B]^\ell \text{ initial block} \\ \bigcup\{RD_{exit}(\ell') \mid \ell' \longrightarrow \ell \text{ in flow chart}\} & \text{otherwise} \end{cases}$$

$$RD_{exit}(\ell) = (RD_{entry}(\ell) \setminus kill_{RD}(B^\ell)) \cup gen_{RD}(B^\ell)$$

where B^ℓ is a block

- A **past** analysis, thus data flows **forwards** (from RD_{entry} to RD_{exit})
- An $\exists$ **path** analysis, thus *lfp* and use $\bigcup$ to merge branches

Some auxiliary definitions

$$\begin{aligned} kill_{RD}([x := a]^\ell) &= \{(x, ?)\} \cup \{(x, \ell') \mid \exists \text{ assignment } [x := \ldots]^{\ell'}\} \\ kill_{RD}([\texttt{skip}]^\ell) &= \emptyset \\ kill_{RD}([b]^\ell) &= \emptyset \end{aligned}$$

$$\begin{aligned} gen_{RD}([x := a]^\ell) &= \{(x, \ell)\} \\ gen_{RD}([\texttt{skip}]^\ell) &= \emptyset \\ gen_{RD}([b]^\ell) &= \emptyset \end{aligned}$$

CONSTRAINT SYSTEMS

Express flow equations in terms of set containments. LV example:

```
[y:=x]1;[z:=1]2; while [y>1]3 do ([z:=z*y]4;[y:=y-1]5);[y:=0]6;
```

$$
\begin{aligned}
LV_{entry}(1) &\supseteq LV_{exit}(1) \setminus \{\mathtt{y}\} \\
LV_{entry}(1) &\supseteq \{\mathtt{x}\} \\
LV_{entry}(2) &\supseteq LV_{exit}(2) \setminus \{\mathtt{z}\} \\
\\
LV_{entry}(3) &\supseteq LV_{exit}(3) \\
LV_{entry}(3) &\supseteq \{\mathtt{y}\} \\
LV_{entry}(4) &\supseteq LV_{exit}(4) \\
LV_{entry}(4) &\supseteq \{\mathtt{y},\mathtt{z}\} \\
LV_{entry}(5) &\supseteq LV_{exit}(5) \\
LV_{entry}(5) &\supseteq \{\mathtt{y}\} \\
LV_{entry}(6) &\supseteq LV_{exit}(6) \setminus \{\mathtt{y}\}
\end{aligned}
$$

$$
\begin{aligned}
LV_{exit}(1) &\supseteq LV_{entry}(2) \\
\\
LV_{exit}(2) &\supseteq LV_{entry}(3) \\
\\
LV_{exit}(3) &\supseteq LV_{entry}(4) \\
LV_{exit}(3) &\supseteq LV_{entry}(6) \\
LV_{exit}(4) &\supseteq LV_{entry}(5) \\
\\
LV_{exit}(5) &\supseteq LV_{entry}(3)
\end{aligned}
$$

Exactly equivalent in this context. More generally: constraints can express more sophisticated flow analyses that are hard to describe by equations.

SEMANTIC CORRECTNESS, OR "SAFETY"

To show: that what the analysis says, is actually true of any computation.

- **Starting point: the semantics of the programming language.**
- **Given a program S and an initial store σ, the semantics defines the set of possible (finite or infinite) computations**

$$\langle S, \sigma \rangle \longrightarrow \langle S_1, \sigma_1 \rangle \longrightarrow \langle S_2, \sigma_2 \rangle \longrightarrow \ldots$$

- **Given: an analysis AN of one (arbitrary) program**
- **Needed: a (logical and natural) connection between**
  - **the result of the analysis; and**
  - **the program's possible computations**

This is the start of the field:

Semantics-based program manipulation