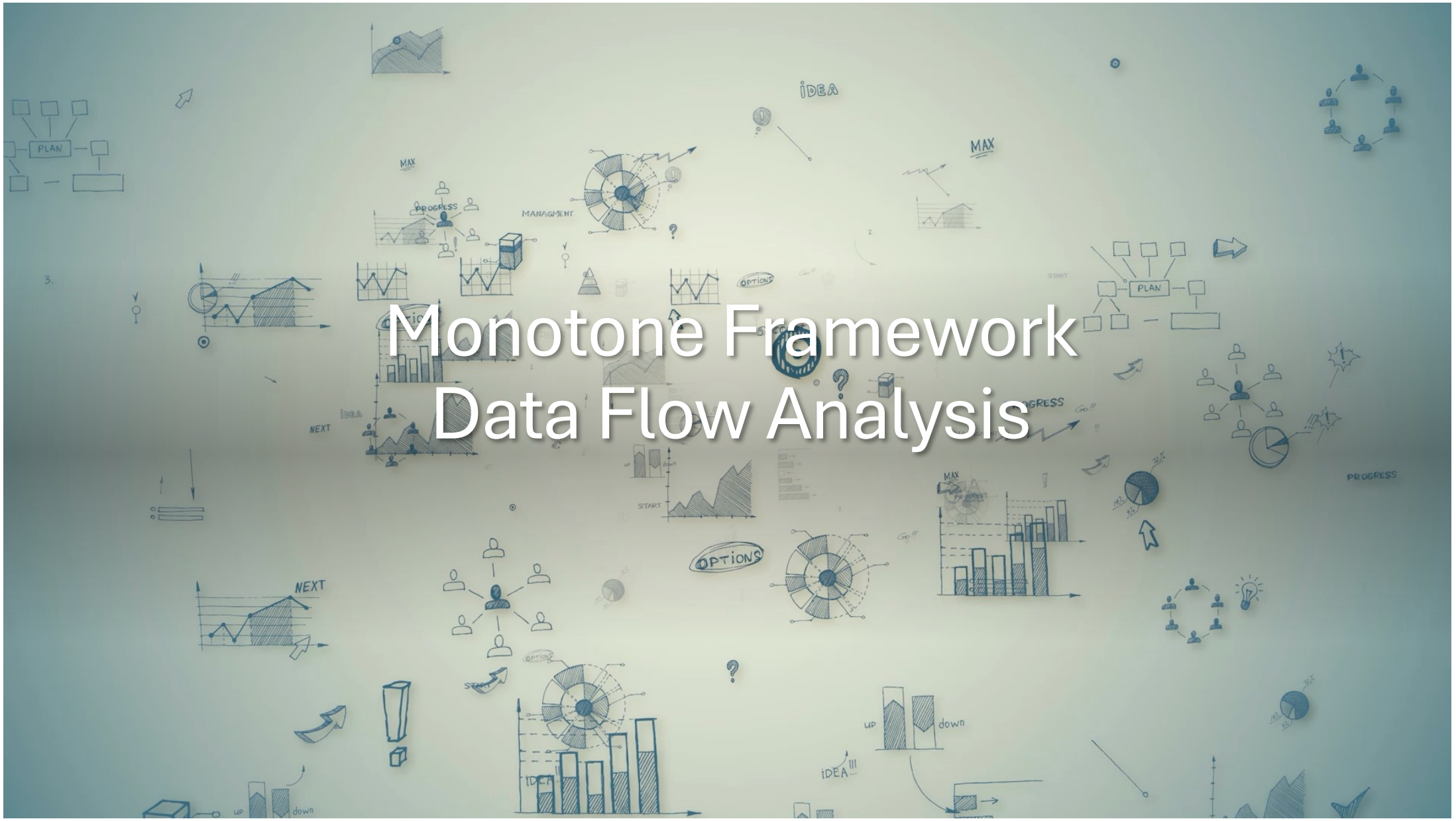

Monotone Framework
Data Flow Analysis

Example of monotone framework in a more complicate analysis: data flow analysis: how data flows inside control flow of program. We would like to have DFA approach but for all sets of execution. We will see how data propagates at

Basic Idea

- Propagate analysis information along the edges of a control flow graph
- Goal: Compute **analysis** state at each program point
- For each statement, define how it affects the analysis state
- For loops: Iterate until fix-point reached

Basic idea of DFA is to propagate how data flow along edges of a CFG. Goal is to compute an analysis state at each program point (the properties we expect at that point of exec.). Because we are working in a monotone fw, we establish constraint and run an iterative algorithm to compute fixed point. One example of Control Flow Analysis is Available Expression Analysis.

Is available expression analysis an example of data flow analysis?

Yes, **available expressions analysis** is a classic example of **data flow analysis**!

Why is it data flow analysis?

Data flow analysis is about tracking how information (like variables, expressions, or properties) propagates through a program's **Control Flow Graph (CFG)**. Available expressions analysis fits this because:

- It computes **which expressions are available** at each program point.
- It follows the **flow of execution** to determine where expressions remain valid.
- It uses **JOIN operations** to merge information from multiple predecessors.

Ex: available expression analysis. ① is what if current value at a program point has been computed before. You can optimize by using value already computed. You avoid repetitive code that does same thing you already did.

Available Expression Analysis

- A (nontrivial) expression is available at a program point if its current value has already been computed earlier in the execution.
- Goal (of the analysis): For each program point, compute which expressions must have already been computed.
- Why?
  - Avoid recomputing an expression
  - Used as part of compiler optimizatization

```
var x = a + b;
var y = a * b;
while (y > a + b) {
  a = a - 1;
  x = a + b;
}
```

```
var x = a + b;
var y = a * b;
while (y > a + b) {
  a = a - 1;
  x = a + b;
}
```

Available every time execution reaches this point

We already computed a+b.

Our approch

We will examine the available expression analysis through two equivalent approaches.

The first, more intuitive, is defined in terms of set theory.

The second is defined in terms of the monotone framework.

- We need two functions: gen (generation): we can define at each point the available expression generated by the statement. Kill eliminates available expression by a statement (ex. point modifies variable value)

Key points

- How the statement affects the analysis state
  - Analysis state = available expressions
- Two functions
  - **gen**: Available expressions generated by a statement
  - **kill**: Available expressions killed by a statement

GEN Set

really a basic block

Power set of non trivial expressions we can have

$$GEN : Stmt \rightarrow \wp(Exp)$$

- The GEN set for a statement (or basic block) contains the expressions that are computed (evaluated) by the statement and whose values are not subsequently invalidated within the same block.
- A statement generates an available expressions e
  - If the statement evaluates e
  - and it does not later write any variable used in e

↓ expressions computed and not modified by basic block.

$$GEN(x = a * b) = \{a * b\}$$
$$GEN(a = a + 1) = \emptyset$$

↓ We evaluate right hands part. a·b is evaluated. a·b is not invalidated (X =) so ok.

POWERSET: set of all possible subsets of a set.

$\{1,2\}$ → POWERSET: $\{1\}, \{2\}, \{1,2\}, \emptyset$

We compute at each program point, the set of available expressions which is a subset of all possible expressions.

- Statement produces an element of the powerset (set of all possible available express.)

So what is Powerset?

GEN IS A FUNCTION WE WILL USE TO DEFINE RULES FOR OUR CONSTRAINTS

KILL Set

set of available expressions

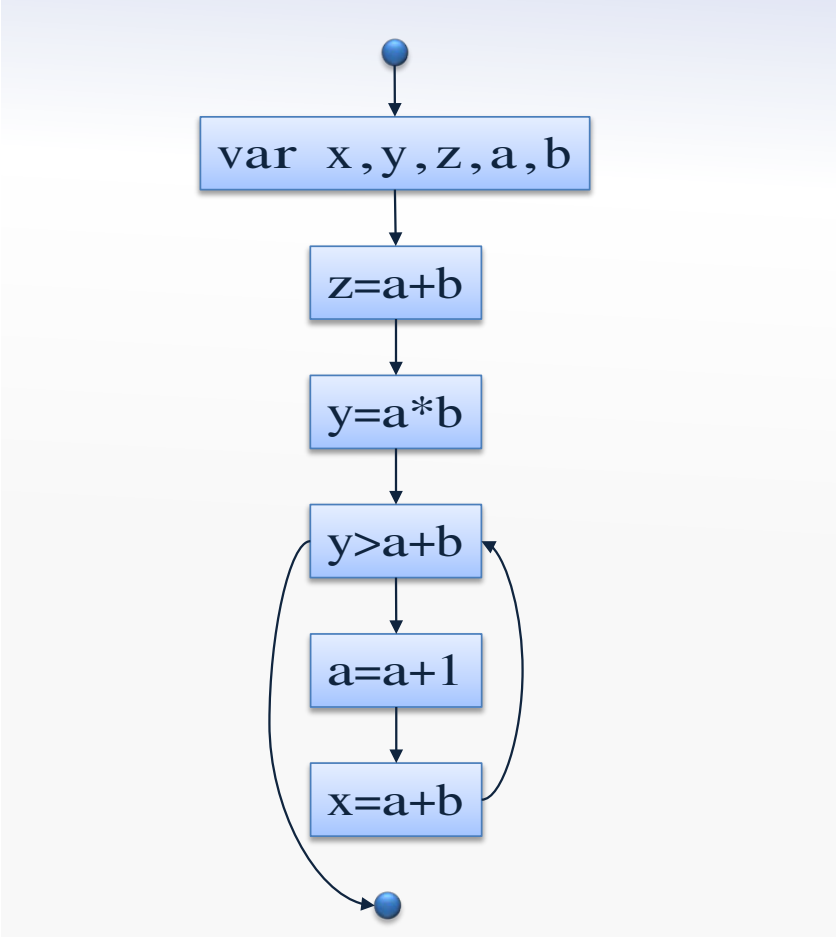
Remove the set whose expressions have been modified and are invalid.

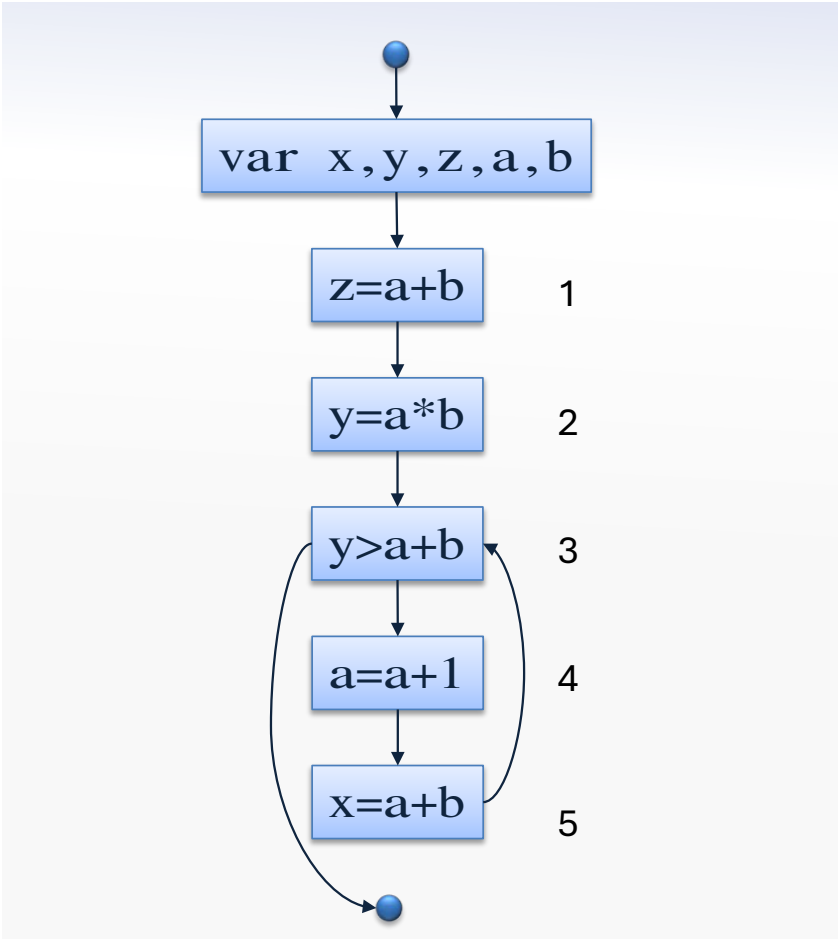
$$\text{KILL}: Stmt \times \wp(Exp) \rightarrow \wp(Exp)$$

- The KILL set for a statement contains expressions that were previously available but are no longer valid because the variables involved in them have been updated. These are available expressions that are invalidated due to reassignments.
- A statement kills an available expressions e
  - if it modifies any of the variables used in e
  - Otherwise returns empty set

$$KILL(a = 5, \{a + b, a * b\}) = \emptyset$$
$$KILL(a = 5, \{b + c, a + c\}) = \{b + c\}$$

```
var x, y, z, a, b;
z= a + b;
y = a * b;
while y > a + b {
   a = a + 1;
   x = a + b;
}
```





Non Trivial Expressions (EXP)
a+b, a*b, a+1

↑ all expressions we have that we consider

STATE	GEN	KILL
1	{a+b}	∅
2	{a*b}	∅
3	{a+b}	∅
4	∅	{a+b, a*b, a+1}
5	{a+b}	∅

update of variable a

for kill

3. y>a+b is a non trivial expression. Why didn't we put it? There's only one point in which it occurs and there's no modification of y after the guard.

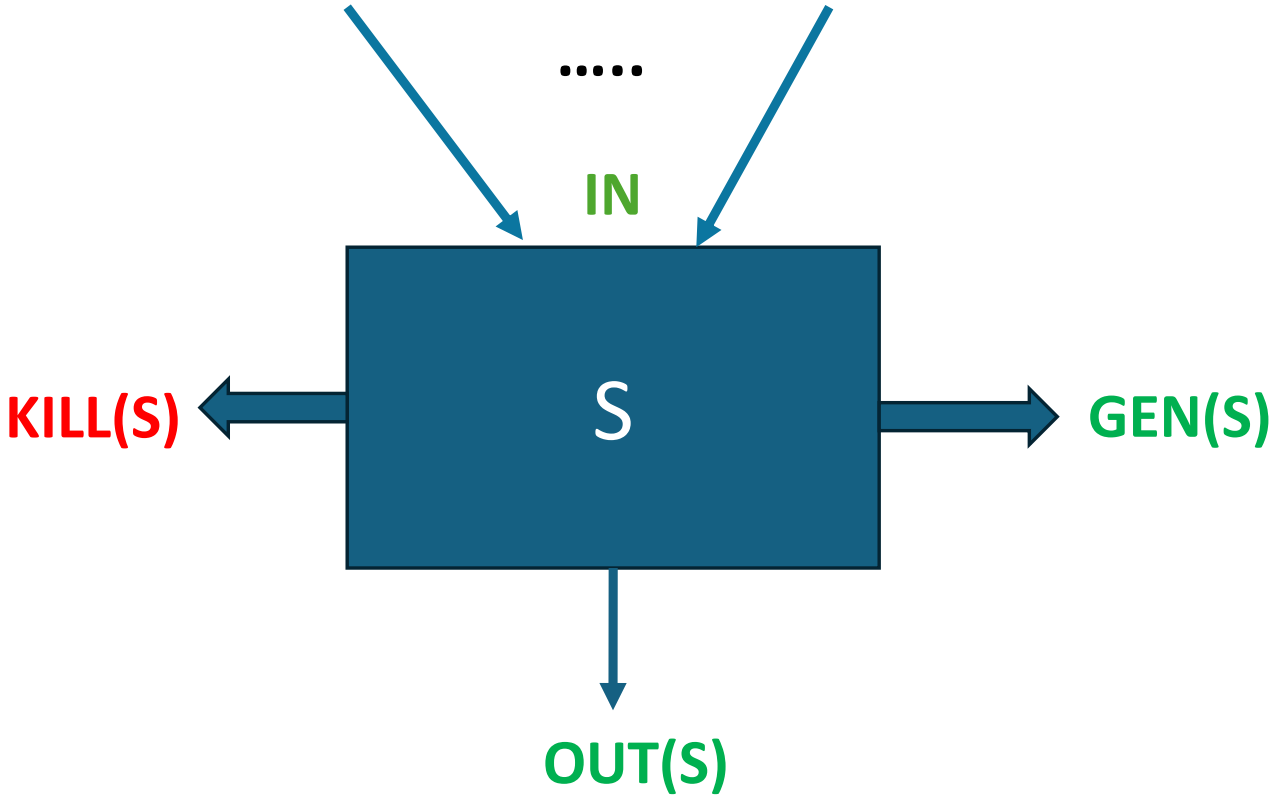
The analysis

Initially no ①. Empty: no info. We follow execution
- Forward analysis: we propagate info we have through node of CFG. For each block we compute two sets: IN: available exp. at beginning of basic block. OUT: available " " end of block.

1. Initially, no available expressions ①
2. Forward analysis: Propagate available expressions in the direction of control flow
3. For each statement s, outgoing available expressions (OUT) :
   - **OUT (s) = (IN(s) - KILL(s, EXP)) ∪ GEN(s) (FLOW EQUATION)** → remove killed expressions and add generated ones.
4. When control flow splits, propagate available expressions both ways
5. When control flows merge, intersect the incoming available expressions

FLOW EQUATION

$$OUT(S) = (IN(S) - KILL(S)) \cup GEN(S)$$



Out(s) = set of available expressions

STATE	GEN	KILL
1	{a+b}	Ø
2	{a*b}	Ø
3	{a+b}	Ø
4	Ø	{a+b, a*b, a+1}
5	{a+b}	Ø

OUT (s) = (IN (s) - KILL(s)) ∪ GEN(s),

IN(1)	Ø
IN(2)	**OUT(1)**
IN(3)	**OUT(2) ∩ OUT(5)**
IN(4)	**OUT(3)**
IN(5)	**OUT4)**
OUT(1)	**IN(1) ∪ {a+b}**
OUT(2)	**IN(2) ∪ {a*b}**
OUT(3)	**IN(3) ∪ {a+b}**
OUT(4)	**IN(4) - {a+b, a*b, a+1}**
OUT(5)	**IN(5)) ∪ {a+b}**

elements coming in are empty

Constraints for nin vs out.

element in to the 2nd node are the ones generated by 1st node.

node associated to condition: we reach it from 2 and loopback from 5:

intersection of the two.

killed

generated

STATE	GEN	KILL
1	{a+b}	∅
2	{a*b}	∅
3	{a+b}	∅
4	∅	{a+b, a*b, a+1}
5	{a+b}	∅

IN(1)	∅
IN(2)	**OUT(1)**
IN(3)	**OUT(2) ∩ OUT(5)**
IN(4)	**OUT(3)**
IN(5)	**OUT4)**
OUT(1)	**IN(1) ∪ {a+b}**
OUT(2)	**IN(2) ∪ {a*b}**
OUT(3)	**IN(3) ∪ {a+b}**
OUT(4)	**IN(4) – {a+b, a*b, a+1}**
OUT(5)	**IN(5)) ∪ {a+b}**

OUT (s) = (IN (s) - KILL(s)) ∪ GEN(s),

ITERATION 1

STATE	IN	OUT
1	∅	∅
2	∅	∅
3	∅	∅
4	∅	∅
5	∅	∅

↗ functions to update in and out

IN and OUT represents info we want

STATE	GEN	KILL
1	{a+b}	∅
2	{a*b}	∅
3	{a+b}	∅
4	∅	{a+b, a*b, a+1}
5	{a+b}	∅

IN(1)	∅
IN(2)	**OUT(1)**
IN(3)	**OUT(2) ∩ OUT(5)**
IN(4)	**OUT(3)**
IN(5)	**OUT4)**
OUT(1)	**IN(1) ∪ {a+b}**
OUT(2)	**IN(2) ∪ {a*b}**
OUT(3)	**IN(3) ∪ {a+b}**
OUT(4)	**IN(4) – {a+b, a*b, a+1}**
OUT(5)	**IN(5)) ∪ {a+b}**

OUT (s) = (IN (s) - KILL(s)) ∪ GEN(s),

ITERATION 1

STATE	IN	OUT
1	∅	∅
2	∅	∅
3	∅	∅
4	∅	∅
5	∅	∅

↳ No info to begin with

ITERATION 2

STATE	IN	OUT
1	∅	$\{a+b\}$
2	∅	$\{a*b\}$
3	∅	$\{a+b\}$
4	∅	∅
5	∅	$\{a+b\}$

STATE	GEN	KILL
1	{a+b}	∅
2	{a*b}	∅
3	{a+b}	∅
4	∅	{a+b, a*b, a+1}
5	{a+b}	∅

IN(1)	∅
IN(2)	**OUT(1)**
IN(3)	**OUT(2) ∩ OUT(5)**
IN(4)	**OUT(3)**
IN(5)	**OUT4)**
OUT(1)	**IN(1) ∪ {a+b}**
OUT(2)	**IN(2) ∪ {a*b}**
OUT(3)	**IN(3) ∪ {a+b}**
OUT(4)	**IN(4) – {a+b, a*b, a+1}**
OUT(5)	**IN(5)) ∪ {a+b}**

OUT (s) = (IN (s) - KILL(s)) ∪ GEN(s),

ITERATION 2

STATE	IN	OUT
1	∅	$\{a+b\}$
2	∅	$\{a*b\}$
3	∅	$\{a+b\}$
4	∅	∅
5	∅	$\{a+b\}$

ITERATION 3

STATE	IN	OUT
1	∅	$\{a+b\}$
2	$\{a+b\}$	$\{a*b\}$
3	∅	$\{a+b\}$
4	$\{a+b\}$	∅
5	∅	$\{a+b\}$

STATE	GEN	KILL
1	{a+b}	∅
2	{a*b}	∅
3	{a+b}	∅
4	∅	{a+b, a*b, a+1}
5	{a+b}	∅

IN(1)	∅
IN(2)	**OUT(1)**
IN(3)	**OUT(2) ∩ OUT(5)**
IN(4)	**OUT(3)**
IN(5)	**OUT4)**
OUT(1)	**IN(1) ∪ {a+b}**
OUT(2)	**IN(2) ∪ {a*b}**
OUT(3)	**IN(3) ∪ {a+b}**
OUT(4)	**IN(4) – {a+b, a*b, a+1}**
OUT(5)	**IN(5)) ∪ {a+b}**

OUT (s) = (IN (s) - KILL(s)) ∪ GEN(s),

ITERATION 3

STATE	IN	OUT
1	∅	$\{a+b\}$
2	$\{a+b\}$	$\{a*b\}$
3	∅	$\{a+b\}$
4	$\{a+b\}$	∅
5	∅	$\{a+b\}$

ITERATION 4

STATE	IN	OUT
1	∅	$\{a+b\}$
2	$\{a+b\}$	$\{a+b, a*b\}$
3	∅	$\{a+b\}$
4	$\{a+b\}$	∅
5	∅	$\{a+b\}$

STATE	GEN	KILL
1	{a+b}	∅
2	{a*b}	∅
3	{a+b}	∅
4	∅	{a+b, a*b, a+1}
5	{a+b}	∅

IN(1)	∅
IN(2)	**OUT(1)**
IN(3)	**OUT(2) ∩ OUT(5)**
IN(4)	**OUT(3)**
IN(5)	**OUT4)**
OUT(1)	**IN(1) ∪ {a+b}**
OUT(2)	**IN(2) ∪ {a*b}**
OUT(3)	**IN(3) ∪ {a+b}**
OUT(4)	**IN(4) – {a+b, a*b, a+1}**
OUT(5)	**IN(5)) ∪ {a+b}**

OUT (s) = (IN (s) - KILL(s)) ∪ GEN(s),

ITERATION 4

STATE	IN	OUT
1	∅	$\{a+b\}$
2	$\{a+b\}$	$\{a+b, a*b\}$
3	∅	$\{a+b\}$
4	$\{a+b\}$	∅
5	∅	$\{a+b\}$

ITERATION 5

STATE	IN	OUT
1	∅	$\{a+b\}$
2	$\{a+b\}$	$\{a+b, a*b\}$
3	~~∅~~ $\{a+b\}$	$\{a+b\}$
4	$\{a+b\}$	∅
5	∅	$\{a+b\}$

```
Var x, y, z, a, b;
1. z = a+b;
2. y = a*b;
3. While (y > a+b) {
4.     a = a+1;
5.     x = a+b;
   }
```

	GEN	KILL
1	{a+b}	∅
2	{a*b}	∅
3	{a+b}	∅
4	∅	{a+b, a+1, a*b}
5	{a+b}	∅

IN(1)	∅
IN(2)	OUT(1)
IN(3)	OUT(2) ∩ OUT(5)
IN(4)	OUT(3)
IN(5)	OUT(4)
OUT(1)	IN(1) ∪ {a+b}
OUT(2)	IN(2) ∪ {a*b}
OUT(3)	IN(3) ∪ {a+b}
OUT(4)	IN(4) - {a+b, a*b, a+1}
OUT(5)	IN(5) ∪ {a+b}

	IN	OUT
1	∅	∅
2	∅	∅
3	∅	∅
4	∅	∅
5	∅	∅

	IN	OUT
1	∅	∅
2	∅	∅
3	∅	∅
4	∅	∅
5	∅	∅

	IN	OUT
1	$\emptyset$	$\{a+b\}$
2	$\emptyset$	$\{a*b\}$
3	$\emptyset$	$\{a+b\}$
4	$\emptyset$	$\emptyset$
5	$\emptyset$	$\{a+b\}$

	IN	OUT
1	$\emptyset$	$\{a+b\}$
2	$\{a+b\}$	$\{a*b, a+b\}$
3	$\emptyset$	$\{a+b\}$
4	$\{a+b\}$	$\emptyset$
5	$\emptyset$	$\{a+b\}$

IN(1)	$\emptyset$
IN(2)	OUT(1)
IN(3)	OUT(2) $\cap$ OUT(5)
IN(4)	OUT(3)
IN(5)	OUT(4)
OUT(1)	IN(1) $\cup \{a+b\}$
OUT(2)	IN(2) $\cup \{a+b\}$
OUT(3)	IN(3) $\cup \{a+b\}$
OUT(4)	IN(4) $- \{a+b, a*b, a+1\}$
OUT(5)	IN(5) $\cup \{a+b\}$

	IN	OUT
1	$\emptyset$	$\{a+b\}$
2	$\{a+b\}$	$\{a*b, a+b\}$
3	$\{a+b\}$	$\{a+b\}$
4	$\{a+b\}$	$\emptyset$
5	$\emptyset$	$\{a+b\}$

STATE	GEN	KILL
1	{a+b}	∅
2	{a*b}	∅
3	{a+b}	∅
4	∅	{a+b, a*b, a+1}
5	{a+b}	∅

IN(1)	∅
IN(2)	**OUT(1)**
IN(3)	**OUT(2) ∩ OUT(5)**
IN(4)	**OUT(3)**
IN(5)	**OUT4)**
OUT(1)	**IN(1) ∪ {a+b}**
OUT(2)	**IN(2) ∪ {a*b}**
OUT(3)	**IN(3) ∪ {a+b}**
OUT(4)	**IN(4) – {a+b, a*b, a+1}**
OUT(5)	**IN(5)) ∪ {a+b}**

OUT (s) = (IN (s) - KILL(s)) ∪ GEN(s),

SOLUTION

STATE	IN	OUT
1	∅	$\{a+b\}$
2	$\{a+b\}$	$\{a+b, a*b\}$
3	∅	$\{a+b\}$
4	$\{a+b\}$	∅
5	∅	$\{a+b\}$

Using set theory we can solve easily

But … what about the monotone framework?

Available Expressions: the lattice

A reverse powerset lattice of nontrivial expressions

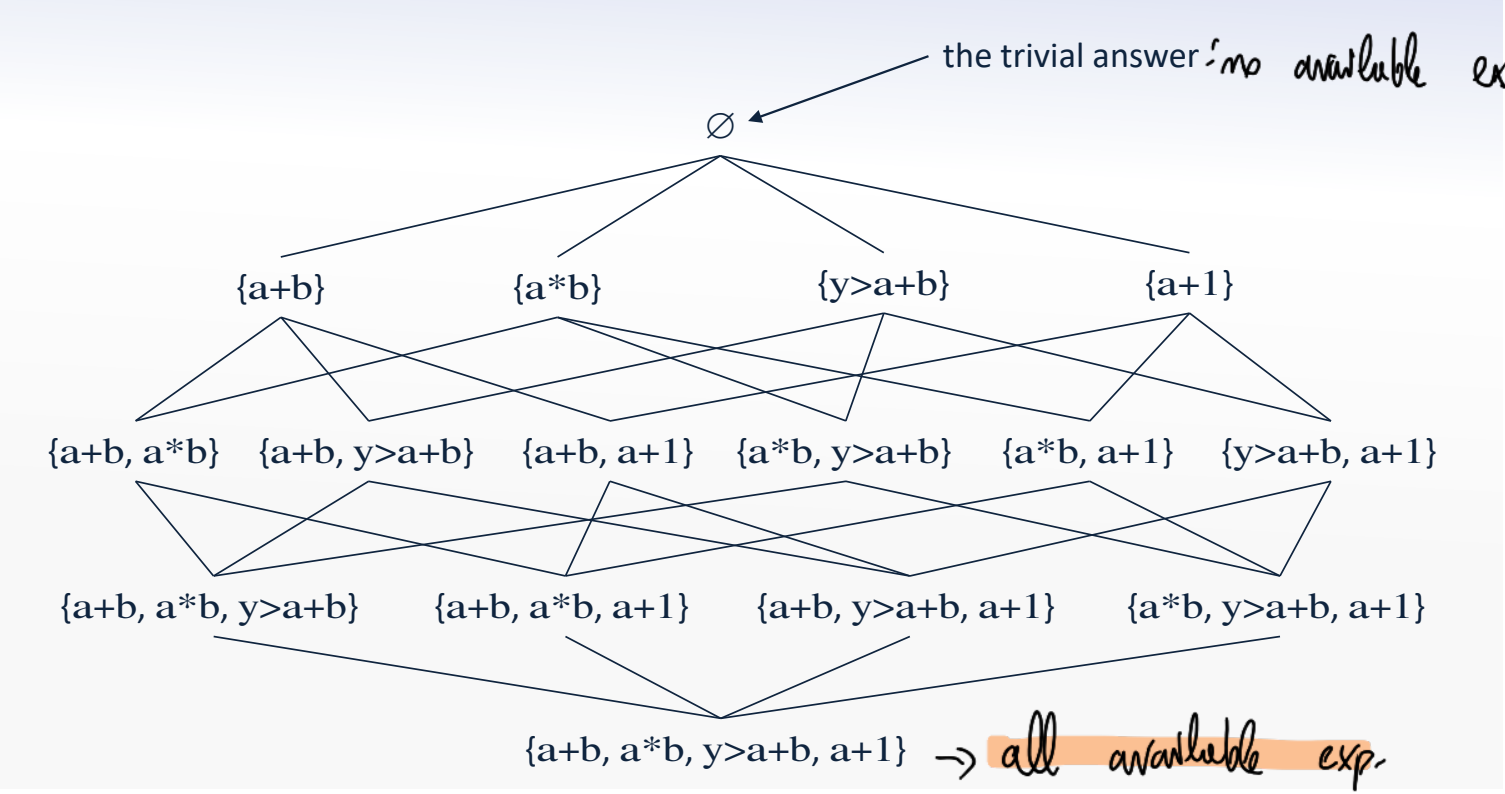
```
var x,y,z,a,b;
z = a+b;
y = a*b;
while (y > a+b) {
   a = a+1;
   x = a+b;
}
```

$L = (\mathcal{P}(\{a+b, a*b, y>a+b, a+1\}), \supseteq)$

Lattice is Powerset of all possible expressions with the order relation $\supseteq$ (reverse of subset inclusion)

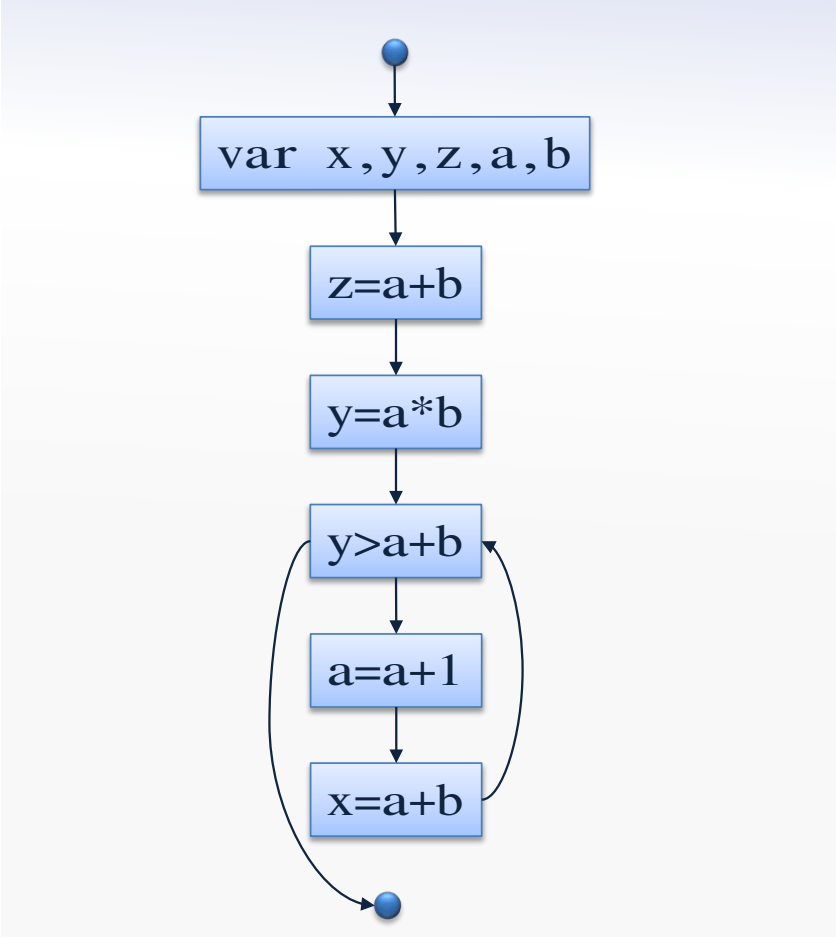
- Standard order in powerset: set is $\leq$ all it is included. We do opposite

The reverse powerset



THE CONTROL FLOW GRAPH

```
var x, y, z, a, b;
z= a + b;
y = a * b;
while y > a + b {
   a = a + 1;
   x = a + b;
}
```



```
var x, y, z, a, b;
z= a + b; 1
y = a * b; 2
while y > a + b {3
  a = a + 1; 4
  x = a + b; 5
}
```

	GEN	KILL
1	$\{a+b\}$	$\emptyset$
2	$\{a*b\}$	$\emptyset$
3	$\{a+b\}$	$\emptyset$
4	$\emptyset$	$\{a+b, a*b, a+1\}$
5	$\{a+b\}$	$\emptyset$

$OUT(S) = (IN(S) - KILL(S,EXP)) \cup GEN(S)$

$IN(1)$	$\emptyset$
$IN(2)$	$OUT(1)$
$IN(3)$	$OUT(2) \cap OUT(5)$
$IN(4)$	$OUT(3)$
$IN(5)$	$OUT(4)$
$OUT(1)$	$IN(1) \cup \{a+b\}$
$OUT(2)$	$IN(2) \cup \{a*b\}$
$OUT(3)$	$IN(3) \cup \{a+b\}$
$OUT(4)$	$IN(4) - \{a+b, a*b, a+1\}$
$OUT(5)$	$IN(5) \cup \{a+b\}$

	IN	OUT
1	$\emptyset$	$\{a+b\}$
2	$\{a+b\}$	$\{a+b, a*b\}$
3	$\emptyset$	$\{a+b\}$
4	$\{a+b\}$	$\emptyset$
5	$\emptyset$	$\{a+b\}$

	IN	OUT
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\emptyset$
3	$\emptyset$	$\emptyset$
4	$\emptyset$	$\emptyset$
5	$\emptyset$	$\emptyset$

	IN	OUT
1	$\emptyset$	$\{a+b\}$
2	$\emptyset$	$\{a*b\}$
3	$\emptyset$	$\{a+b\}$
4	$\emptyset$	$\emptyset$
5	$\emptyset$	$\{a+b\}$

	IN	OUT
1	$\emptyset$	$\{a+b\}$
2	$\{a+b\}$	$\{a*b\}$
3	$\emptyset$	$\{a+b\}$
4	$\{a+b\}$	$\emptyset$
5	$\emptyset$	$\{a+b\}$

	IN	OUT
1	$\emptyset$	$\{a+b\}$
2	$\{a+b\}$	$\{a+b, a*b\}$
3	$\{a+b\}$	$\{a+b\}$
4	$\{a+b\}$	$\emptyset$
5	$\emptyset$	$\{a+b\}$

On each iteration, you switch between changing IN and changing OUT.

```
var x, y, z, a, b;
z= a + b;
y = a * b;
while y > a + b {
  a = a + 1;
  x = a + b;
}
```

$[start] = \emptyset$

$[var\ x, y, z, a, b] = [start]$

$[z = a+b] = [var\ x, y, z, a, b] \cup exps(a+b) \downarrow z$

$[y = a*b] = [z = a+b] \cup exps(a*b) \downarrow y$

$[y > a+b] = ([y = a*b] \cap [x = a+b]) \cup exps(y > a+b)$

$[a = a+1] = [y > a+b] \cup exps(a+1) \downarrow a$

$[x = a+b] = [a = a+1] \cup exps(a+b) \downarrow x$

$[exit] = [y > a+b]$

Setting up the monotone framework

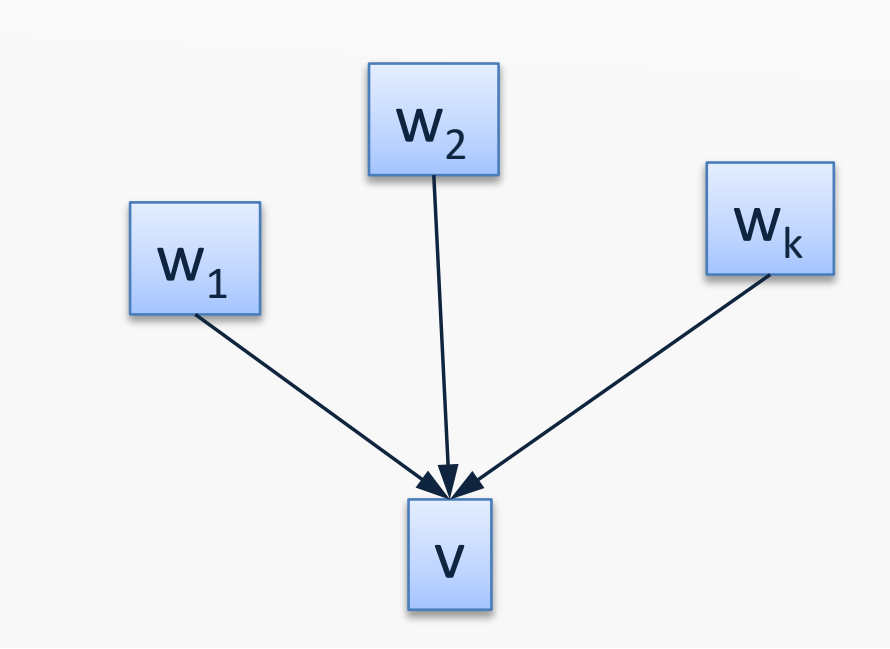
For every node v of the control flow graph we have a variable

$[\![v]\!]$ control flow framework.

yielding the set of expressions that are avaialable at the program point after v

for i assowates from any program point s an element of the lattice

JOIN (revisited)



We take predecessors and do least upper bounds. Here we take the intersection in this case.

$$JOIN(v) = \bigcap_{w \in pred(v)} [\![w]\!]$$

Join in our reverse order

Auxiliary definitions

$$S \subset \wp(Exp)$$

Powerset

$$S \downarrow x = S - \{E \in Exp \mid x \in Var(E)\}$$

function taking a set and producing S - all expressions where Var x occurs in - expressions, Vars as the kill

The function S↓x removes all expressions that contain the variable x from the set S

Auxiliary definitions

The function $exps(E)$ is defined as follows

for constant or variable no expressions

$$exps(CnstInt) = \emptyset$$

$$exps(x) = \emptyset \rightarrow \text{no expr. inside exp.}$$

$$exps(e_1 \, op \, e_2) = \{e_1 \, op \, e_2\} \cup exps(e_1) \cup exps(e_2)$$

↳ denoting

Intuition: The function $exps(E)$ returns the set of non-trivial expressions that make up the expression E.

available exp. associated to e_1 op e_2.

$a + (a+b) \rightarrow \{a + (a+b)\} \cup \emptyset \cup \{a+b\} \cup \emptyset$

We denote available expressions from E with this.

The constraints

- For the *entry* node:

  $[\![entry]\!] = \emptyset$ No constraints

- For conditions and output:

  $[\![\mathbf{if}\ (E)]\!] = [\![\mathbf{output}\ E]\!] = JOIN(v) \cup exps(E)$

  set of available expressions

- For assignments:

  $[\![x = E]\!] = (JOIN(v) \cup exps(E))\downarrow x$

  we take all the IN by join

  take value by expression E and kill E.

- For any other node v:

  $[\![v]\!] = JOIN(v)$

The constraints

- For the *entry* node:

  $[\![entry]\!] = \varnothing$

- For conditions and output:

  $[\![\mathbf{if}\ \ (E)]\!] = [\![\mathbf{output}\ E]\!] = JOIN(\mathrm{v}) \cup exps(E)$

- For assignments:

  $[\![x = E]\!] = (JOIN(\mathrm{v}) \cup exps(E)) \downarrow x$

- For any other node v:

  $[\![\mathrm{v}]\!] = JOIN(\mathrm{v})$

Equations in terms of OPs: RHSs is the answer

printing (output)

FLOW EQUATION

The generated constraints

$[\![entry]\!] = \varnothing$

$[\![\mathrm{var}\ \ x,y,z,a,b]\!] = [\![entry]\!]$

$[\![z=a+b]\!] = exps(a+b)\downarrow z$

$[\![y=a*b]\!] = ([\![z=a+b]\!] \cup exps(a*b))\downarrow y$

$[\![y>a+b]\!] = ([\![y=a*b]\!] \cap [\![x=a+b]\!]) \cup exps(y>a+b)$

$[\![a=a+1]\!] = ([\![y>a+b]\!] \cup exps(a+1))\downarrow a$

$[\![x=a+b]\!] = ([\![a=a+1]\!] \cup exps(a+b))\downarrow x$

$[\![exit]\!] = [\![y>a+b]\!]$

The least solution

You get these by starting with bottoms as inputs.

$[\![\mathit{entry}]\!] = \varnothing$

$[\![\mathrm{var}\ \ x,y,z,a,b]\!] = \varnothing$

$[\![z=a+b]\!] = \{a+b\}$

$[\![y=a*b]\!] = \{a+b,\ a*b\}$

$[\![y>a+b]\!] = \{a+b,\ y>a+b\}$

$[\![a=a+1]\!] = \varnothing$

$[\![x=a+b]\!] = \{a+b\}$

$[\![\mathit{exit}]\!] = \{a+b\}$

- The program can be optimized (slightly):

```
var x,y,x,a,b,aplusb;
aplusb = a+b;
z = aplusb;
y = a*b;
while (y > aplusb) {
  a = a+1;
  aplusb = a+b;
  x = aplusb;
}
```