L18: DFA: Very Busy, Available, and Live

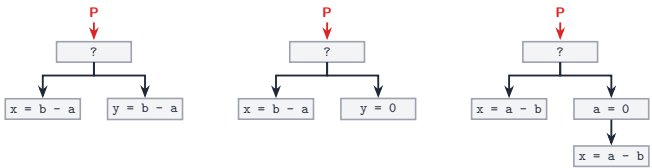
Software Analysis

IIT Guwahati · September 9, 2026

Very Busy Expression Analysis

What Makes an Expression Very Busy?

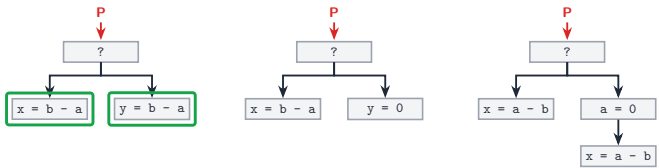
Two things must hold on *every* path leaving the point: the expression is **used**, and it is used **before** any of its variables changes.



Very Busy Expression Analysis

What Makes an Expression Very Busy?

Two things must hold on *every* path leaving the point: the expression is **used**, and it is used **before** any of its variables changes.

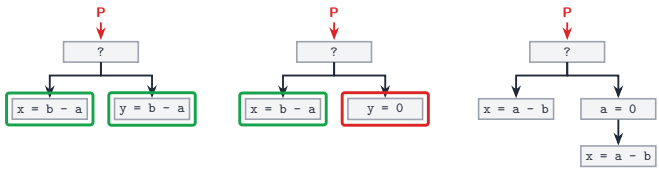


very busy

Very Busy Expression Analysis

What Makes an Expression Very Busy?

Two things must hold on *every* path leaving the point: the expression is **used**, and it is used **before** any of its variables changes.

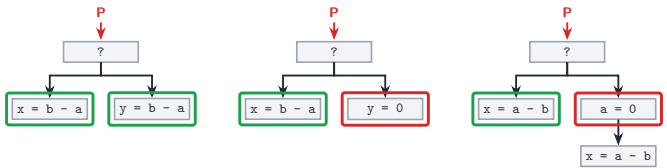


very busy **not used here**

Very Busy Expression Analysis

What Makes an Expression Very Busy?

Two things must hold on *every* path leaving the point: the expression is **used**, and it is used **before** any of its variables changes.

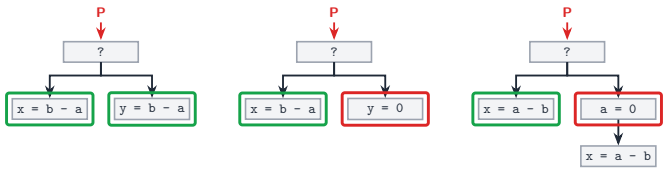


very busy **not used here** a **changes first**

Very Busy Expression Analysis

What Makes an Expression Very Busy?

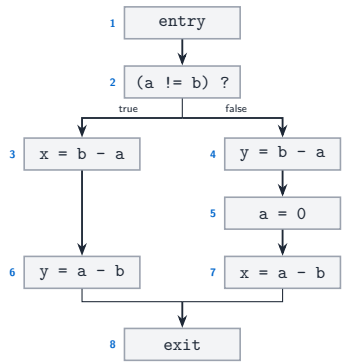
Two things must hold on *every* path leaving the point: the expression is **used**, and it is used **before** any of its variables changes.



Only the first is very busy at **P**. In the third, `a - b` *is* recomputed later: it is a *different* value.

Goal: Very Busy Expressions

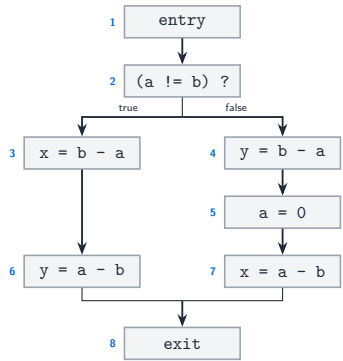
An expression is **very busy** at a program point if, *no matter which path is taken*, it is always used before any of the variables occurring in it is redefined.



Goal: Very Busy Expressions

An expression is **very busy** at a program point if, *no matter which path is taken*, it is always used before any of the variables occurring in it is redefined.

Two expressions here: a − b and b − a.

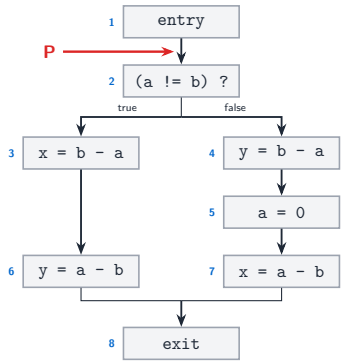


Goal: Very Busy Expressions

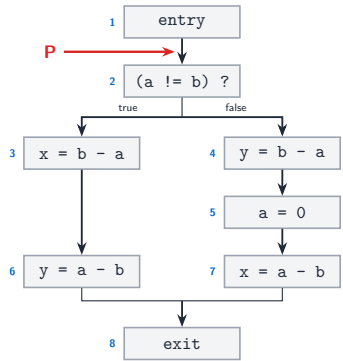
An expression is **very busy** at a program point if, *no matter which path is taken*, it is always used before any of the variables occurring in it is redefined.

Two expressions here: $a - b$ and $b - a$.

Consider point **P**, the entry of node 2.



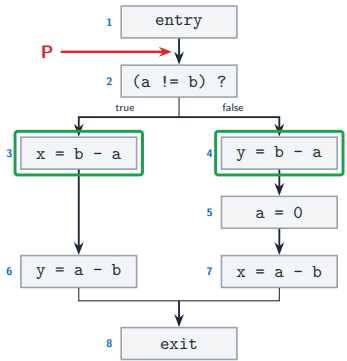
One Is Very Busy, One Is Not



One Is Very Busy, One Is Not

`b − a` is very busy at P.

Used at node 3 on the true path,
used at node 4 on the false path.
Neither `a` nor `b` changes first.



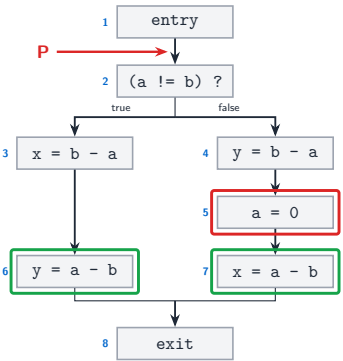
One Is Very Busy, One Is Not

b − a is very busy at P.

Used at node 3 on the true path,
used at node 4 on the false path.
Neither a nor b changes first.

a − b is *not* very busy at P.

Used at node 6 on the true path ✓
but on the false path node 5 sets a = 0
before node 7 uses it. ×



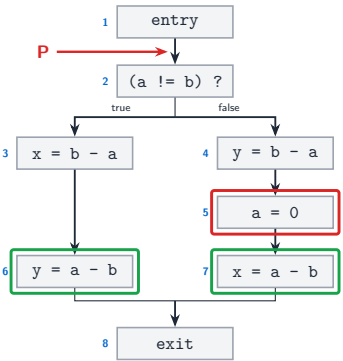
One Is Very Busy, One Is Not

b − a is very busy at P.

Used at node 3 on the true path,
used at node 4 on the false path.
Neither a nor b changes first.

a − b is *not* very busy at P.

Used at node 6 on the true path ✓
but on the false path node 5 sets a = 0
before node 7 uses it. ×



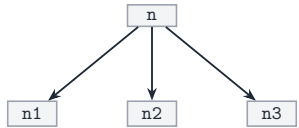
One bad path is enough to disqualify an expression — that is what **must** means.

Operation #1: Merge at the Exit

$$\mathrm{OUT}[n] = \bigcap_{n' \in succ(n)} \mathrm{IN}[n']$$

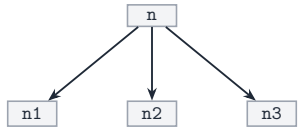
Operation #1: Merge at the Exit

$$\mathrm{OUT}[n] = \bigcap_{n' \in succ(n)} \mathrm{IN}[n']$$



Operation #1: Merge at the Exit

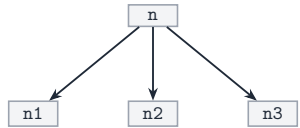
$$\text{OUT}[n] = \bigcap_{n' \in succ(n)} \text{IN}[n']$$



$$\text{OUT}[n] = \text{IN}[\texttt{n1}] \cap \text{IN}[\texttt{n2}] \cap \text{IN}[\texttt{n3}]$$

Operation #1: Merge at the Exit

$$\text{OUT}[n] = \bigcap_{n' \in succ(n)} \text{IN}[n']$$



$$\text{OUT}[n] = \text{IN}[\texttt{n1}] \cap \text{IN}[\texttt{n2}] \cap \text{IN}[\texttt{n3}]$$

Two changes from reaching definitions: **successors** not predecessors, and $\cap$ not $\cup$.

Operation #2: Transfer Through the Node

$$\text{IN}[n] \;=\; \big(\text{OUT}[n] - \textbf{KILL}[n]\big) \;\cup\; \textbf{GEN}[n]$$

Operation #2: Transfer Through the Node

$$\text{IN}[n] \;=\; \big(\text{OUT}[n] - \textbf{KILL}[n]\big) \;\cup\; \textbf{GEN}[n]$$

- **GEN**[n] — expressions *used* by node n.

Operation #2: Transfer Through the Node

$$\text{IN}[n] \;=\; \big(\text{OUT}[n] - \textbf{KILL}[n]\big) \;\cup\; \textbf{GEN}[n]$$

- **GEN**[n] — expressions *used* by node n.
- **KILL**[n] — expressions whose variables node n *modifies*.

Operation #2: Transfer Through the Node

$$\text{IN}[n] \;=\; \big(\text{OUT}[n] - \textbf{KILL}[n]\big) \;\cup\; \textbf{GEN}[n]$$

- **GEN**[n] — expressions *used* by node n.
- **KILL**[n] — expressions whose variables node n *modifies*.
- Same shape as before, with **IN and OUT swapped**.

Very Busy: the GEN and KILL Rules

Statement at n	**GEN**$[n]$	**KILL**$[n]$
a condition, e.g. $(\mathtt{a} != \mathtt{b})$		
an assignment $\mathtt{x} = \mathtt{a}$		

Very Busy: the GEN and KILL Rules

Statement at n	**GEN**$[n]$	**KILL**$[n]$
a condition, e.g. (a! = b)	$\emptyset$	$\emptyset$
an assignment x = a		

Very Busy: the GEN and KILL Rules

Statement at n	**GEN**$[n]$	**KILL**$[n]$
a condition, e.g. (a! = b)	$\emptyset$	$\emptyset$
an assignment x = a	$\{$ a $\}$	every expression using x

Very Busy: the GEN and KILL Rules

Statement at n	**GEN**$[n]$	**KILL**$[n]$
a condition, e.g. (a! = b)	$\emptyset$	$\emptyset$
an assignment x = a	{ a }	every expression using x

A condition modifies and generates nothing: $\mathrm{IN}[n] = \mathrm{OUT}[n]$.

Very Busy: the GEN and KILL Rules

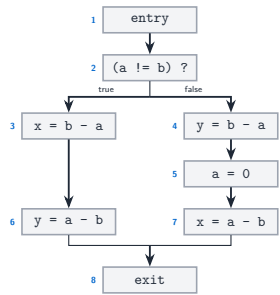
Statement at n	GEN$[n]$	KILL$[n]$
a condition, e.g. (a! = b)	$\emptyset$	$\emptyset$
an assignment x = a	{ a }	every expression using x

A condition modifies and generates nothing: $\mathrm{IN}[n] = \mathrm{OUT}[n]$.

An assignment **uses** its right-hand side and **invalidates** everything mentioning its left-hand side.

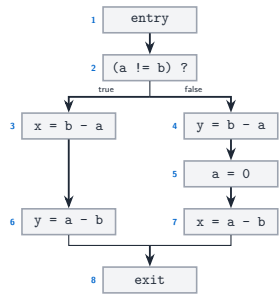
Very Busy: GEN and KILL for the Program

n	Statement	GEN	KILL
2	$(a != b)?$		
3	$x = b - a$		
4	$y = b - a$		
5	$a = 0$		
6	$y = a - b$		
7	$x = a - b$		



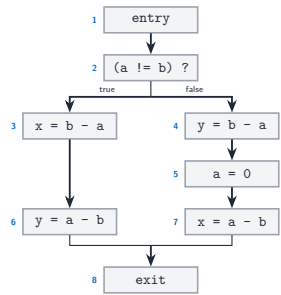
Very Busy: GEN and KILL for the Program

n	Statement	GEN	KILL
2	$(a! = b)?$	$\emptyset$	$\emptyset$
3	$x = b - a$		
4	$y = b - a$		
5	$a = 0$		
6	$y = a - b$		
7	$x = a - b$		



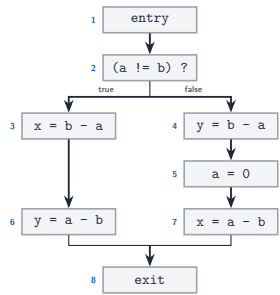
Very Busy: GEN and KILL for the Program

n	Statement	GEN	KILL
2	$(a != b)?$	$\emptyset$	$\emptyset$
3	$x = b - a$	$\{b - a\}$	$\emptyset$
4	$y = b - a$		
5	$a = 0$		
6	$y = a - b$		
7	$x = a - b$		



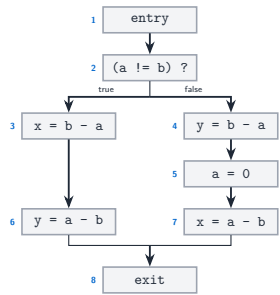
Very Busy: GEN and KILL for the Program

n	Statement	GEN	KILL
2	$(a != b)?$	$\emptyset$	$\emptyset$
3	$x = b - a$	$\{b - a\}$	$\emptyset$
4	$y = b - a$	$\{b - a\}$	$\emptyset$
5	$a = 0$		
6	$y = a - b$		
7	$x = a - b$		



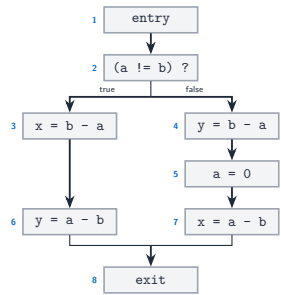
Very Busy: GEN and KILL for the Program

n	Statement	GEN	KILL
2	$(a! = b)?$	$\emptyset$	$\emptyset$
3	$x = b - a$	$\{b - a\}$	$\emptyset$
4	$y = b - a$	$\{b - a\}$	$\emptyset$
5	$a = 0$	$\emptyset$	$\{a - b, b - a\}$
6	$y = a - b$		
7	$x = a - b$		



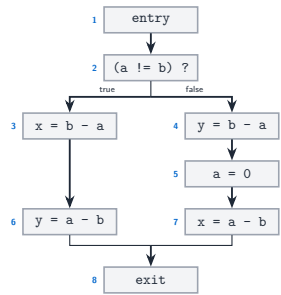
Very Busy: GEN and KILL for the Program

n	Statement	GEN	KILL
2	$(a != b)?$	$\emptyset$	$\emptyset$
3	$x = b - a$	$\{b - a\}$	$\emptyset$
4	$y = b - a$	$\{b - a\}$	$\emptyset$
5	$a = 0$	$\emptyset$	$\{a - b, b - a\}$
6	$y = a - b$	$\{a - b\}$	$\emptyset$
7	$x = a - b$		



Very Busy: GEN and KILL for the Program

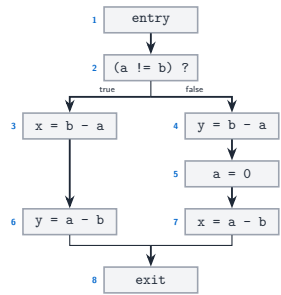
n	Statement	GEN	KILL
2	$(a != b)?$	$\emptyset$	$\emptyset$
3	$x = b - a$	$\{b - a\}$	$\emptyset$
4	$y = b - a$	$\{b - a\}$	$\emptyset$
5	$a = 0$	$\emptyset$	$\{a - b, b - a\}$
6	$y = a - b$	$\{a - b\}$	$\emptyset$
7	$x = a - b$	$\{a - b\}$	$\emptyset$



Very Busy: GEN and KILL for the Program

n	Statement	GEN	KILL
2	$(\mathtt{a} != \mathtt{b})?$	$\emptyset$	$\emptyset$
3	$\mathtt{x} = \mathtt{b} - \mathtt{a}$	$\{\mathtt{b} - \mathtt{a}\}$	$\emptyset$
4	$\mathtt{y} = \mathtt{b} - \mathtt{a}$	$\{\mathtt{b} - \mathtt{a}\}$	$\emptyset$
5	$\mathtt{a} = 0$	$\emptyset$	$\{\mathtt{a} - \mathtt{b}, \mathtt{b} - \mathtt{a}\}$
6	$\mathtt{y} = \mathtt{a} - \mathtt{b}$	$\{\mathtt{a} - \mathtt{b}\}$	$\emptyset$
7	$\mathtt{x} = \mathtt{a} - \mathtt{b}$	$\{\mathtt{a} - \mathtt{b}\}$	$\emptyset$

Only node 5 kills anything — and it kills *both*, since a occurs in both expressions.



Three Differences from Reaching Definitions

- **The roles of IN and OUT are swapped.**

  Information propagates backwards.

Three Differences from Reaching Definitions

- **The roles of IN and OUT are swapped.**

  Information propagates backwards.

- **Intersection, not union.**

  So the sets *shrink* as the algorithm runs, instead of growing.

Three Differences from Reaching Definitions

- **The roles of IN and OUT are swapped.**

  Information propagates backwards.

- **Intersection, not union.**

  So the sets *shrink* as the algorithm runs, instead of growing.

- **Initialise to the set of *all* expressions.**

  Sets shrink, so they must start at the top. Boundary: $\mathrm{IN}[\mathit{exit}] = \emptyset$.

Three Differences from Reaching Definitions

- **The roles of IN and OUT are swapped.**

  Information propagates backwards.

- **Intersection, not union.**

  So the sets *shrink* as the algorithm runs, instead of growing.

- **Initialise to the set of *all* expressions.**

  Sets shrink, so they must start at the top. Boundary: $\mathrm{IN}[\mathit{exit}] = \emptyset$.

Very Busy: the Abstract Domain

- Expressions: $\texttt{a} - \texttt{b}$ and $\texttt{b} - \texttt{a}$.

Very Busy: the Abstract Domain

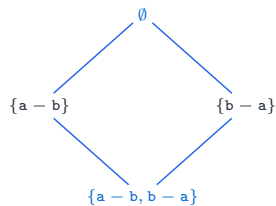
- Expressions: `a − b` and `b − a`.
- Any *combination* of them may be very busy, so each combination is an **abstract value**.

Very Busy: the Abstract Domain

- Expressions: `a - b` and `b - a`.
- Any *combination* of them may be very busy, so each combination is an **abstract value**.
- The **abstract domain** is the powerset of the *expressions*: $2^2 = 4$ values.

Very Busy: the Abstract Domain

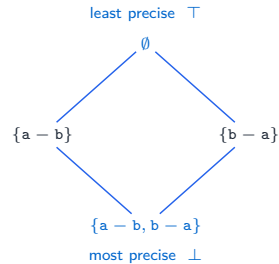
- Expressions: $\texttt{a} - \texttt{b}$ and $\texttt{b} - \texttt{a}$.
- Any *combination* of them may be very busy, so each combination is an **abstract value**.
- The **abstract domain** is the powerset of the *expressions*: $2^2 = 4$ values.
- Ordered by **reverse** set inclusion.



Very Busy: the Abstract Domain

- Expressions: $\mathtt{a} - \mathtt{b}$ and $\mathtt{b} - \mathtt{a}$.
- Any *combination* of them may be very busy, so each combination is an **abstract value**.
- The **abstract domain** is the powerset of the *expressions*: $2^2 = 4$ values.
- Ordered by **reverse** set inclusion.

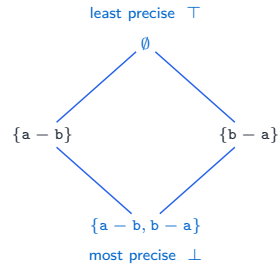
A *larger* set of very busy expressions is a **more precise** result.



Very Busy: the Abstract Domain

- Expressions: $\mathtt{a} - \mathtt{b}$ and $\mathtt{b} - \mathtt{a}$.
- Any *combination* of them may be very busy, so each combination is an **abstract value**.
- The **abstract domain** is the powerset of the *expressions*: $2^2 = 4$ values.
- Ordered by **reverse** set inclusion.

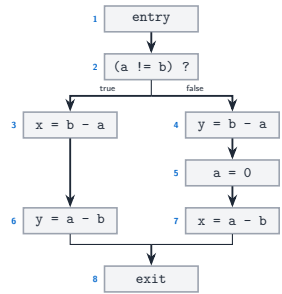
A *larger* set of very busy expressions is a **more precise** result.



Very Busy: Initialisation

Every set starts as $\{a - b, b - a\}$, all the expressions.

n	**IN**$[n]$	**OUT**$[n]$
1	—	$\{a - b, b - a\}$
2	$\{a - b, b - a\}$	$\{a - b, b - a\}$
3	$\{a - b, b - a\}$	$\{a - b, b - a\}$
4	$\{a - b, b - a\}$	$\{a - b, b - a\}$
5	$\{a - b, b - a\}$	$\{a - b, b - a\}$
6	$\{a - b, b - a\}$	$\{a - b, b - a\}$
7	$\{a - b, b - a\}$	$\{a - b, b - a\}$
8		—

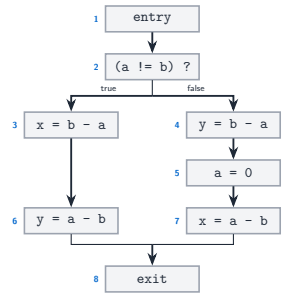


Very Busy: Initialisation

Every set starts as $\{a - b, b - a\}$, all the expressions.

n	**IN**[n]	**OUT**[n]
1	—	$\{a - b, b - a\}$
2	$\{a - b, b - a\}$	$\{a - b, b - a\}$
3	$\{a - b, b - a\}$	$\{a - b, b - a\}$
4	$\{a - b, b - a\}$	$\{a - b, b - a\}$
5	$\{a - b, b - a\}$	$\{a - b, b - a\}$
6	$\{a - b, b - a\}$	$\{a - b, b - a\}$
7	$\{a - b, b - a\}$	$\{a - b, b - a\}$
8	$\emptyset$	—

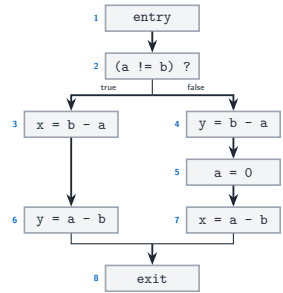
Only IN[8] differs: nothing is very busy at the end.



Very Busy: Pass 1

Visiting nodes $7, 6, 5, 4, 3, 2, 1$.

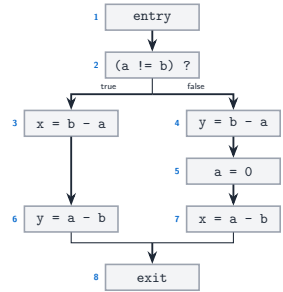
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\{a - b, b - a\}$
2	$\{a - b, b - a\}$	$\{a - b, b - a\}$
3	$\{a - b, b - a\}$	$\{a - b, b - a\}$
4	$\{a - b, b - a\}$	$\{a - b, b - a\}$
5	$\{a - b, b - a\}$	$\{a - b, b - a\}$
6	$\{a - b, b - a\}$	$\{a - b, b - a\}$
7	$\{a - b\}$	$\emptyset$
8	$\emptyset$	—



Very Busy: Pass 1

Visiting nodes $7, 6, 5, 4, 3, 2, 1$.

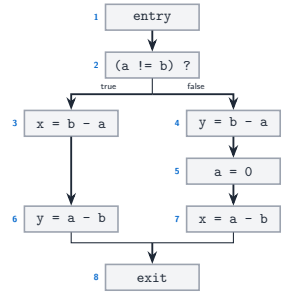
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\{a - b, b - a\}$
2	$\{a - b, b - a\}$	$\{a - b, b - a\}$
3	$\{a - b, b - a\}$	$\{a - b, b - a\}$
4	$\{a - b, b - a\}$	$\{a - b, b - a\}$
5	$\{a - b, b - a\}$	$\{a - b, b - a\}$
6	$\{a - b\}$	$\emptyset$
7	$\{a - b\}$	$\emptyset$
8	$\emptyset$	—



Very Busy: Pass 1

Visiting nodes $7, 6, 5, 4, 3, 2, 1$.

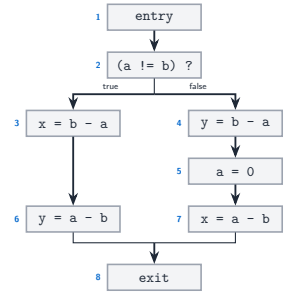
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\{a - b, b - a\}$
2	$\{a - b, b - a\}$	$\{a - b, b - a\}$
3	$\{a - b, b - a\}$	$\{a - b, b - a\}$
4	$\{a - b, b - a\}$	$\{a - b, b - a\}$
5	$\emptyset$	$\{a - b\}$
6	$\{a - b\}$	$\emptyset$
7	$\{a - b\}$	$\emptyset$
8	$\emptyset$	—



Very Busy: Pass 1

Visiting nodes $7, 6, 5, 4, 3, 2, 1$.

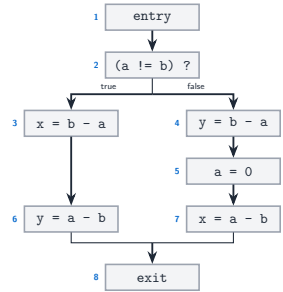
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\{a - b, b - a\}$
2	$\{a - b, b - a\}$	$\{a - b, b - a\}$
3	$\{a - b, b - a\}$	$\{a - b, b - a\}$
4	$\{b - a\}$	$\emptyset$
5	$\emptyset$	$\{a - b\}$
6	$\{a - b\}$	$\emptyset$
7	$\{a - b\}$	$\emptyset$
8	$\emptyset$	—



Very Busy: Pass 1

Visiting nodes $7, 6, 5, 4, 3, 2, 1$.

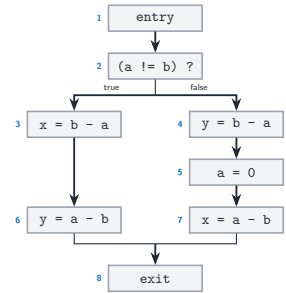
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\{a - b, b - a\}$
2	$\{a - b, b - a\}$	$\{a - b, b - a\}$
3	$\{a - b, b - a\}$	$\{a - b\}$
4	$\{b - a\}$	$\emptyset$
5	$\emptyset$	$\{a - b\}$
6	$\{a - b\}$	$\emptyset$
7	$\{a - b\}$	$\emptyset$
8	$\emptyset$	—



Very Busy: Pass 1

Visiting nodes $7, 6, 5, 4, 3, 2, 1$.

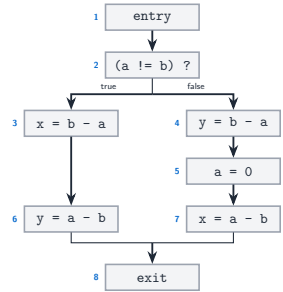
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\{a - b, b - a\}$
2	$\{b - a\}$	$\{b - a\}$
3	$\{a - b, b - a\}$	$\{a - b\}$
4	$\{b - a\}$	$\emptyset$
5	$\emptyset$	$\{a - b\}$
6	$\{a - b\}$	$\emptyset$
7	$\{a - b\}$	$\emptyset$
8	$\emptyset$	—



Very Busy: Pass 1

Visiting nodes $7, 6, 5, 4, 3, 2, 1$.

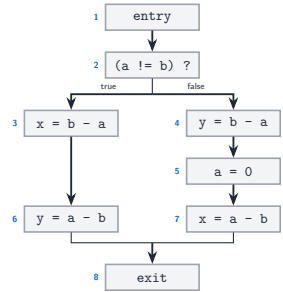
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\{b - a\}$
2	$\{b - a\}$	$\{b - a\}$
3	$\{a - b, b - a\}$	$\{a - b\}$
4	$\{b - a\}$	$\emptyset$
5	$\emptyset$	$\{a - b\}$
6	$\{a - b\}$	$\emptyset$
7	$\{a - b\}$	$\emptyset$
8	$\emptyset$	—



Very Busy: Pass 1

Visiting nodes $7, 6, 5, 4, 3, 2, 1$.

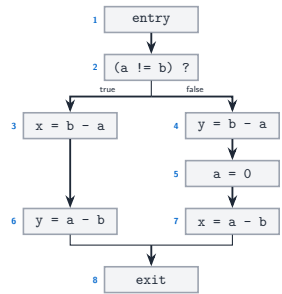
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\{b - a\}$
2	$\{b - a\}$	$\{b - a\}$
3	$\{a - b, b - a\}$	$\{a - b\}$
4	$\{b - a\}$	$\emptyset$
5	$\emptyset$	$\{a - b\}$
6	$\{a - b\}$	$\emptyset$
7	$\{a - b\}$	$\emptyset$
8	$\emptyset$	—



Node 5 wipes both expressions, so $a - b$ never escapes the false branch.

Very Busy: Pass 2 — Saturated

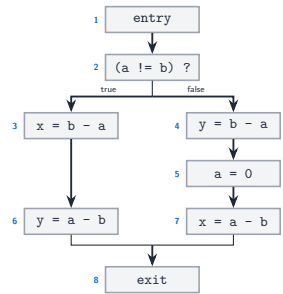
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\{b - a\}$
2	$\{b - a\}$	$\{b - a\}$
3	$\{a - b, b - a\}$	$\{a - b\}$
4	$\{b - a\}$	$\emptyset$
5	$\emptyset$	$\{a - b\}$
6	$\{a - b\}$	$\emptyset$
7	$\{a - b\}$	$\emptyset$
8	$\emptyset$	—



Very Busy: Pass 2 — Saturated

n	**IN**[n]	**OUT**[n]
1	—	$\{b - a\}$
2	$\{b - a\}$	$\{b - a\}$
3	$\{a - b, b - a\}$	$\{a - b\}$
4	$\{b - a\}$	$\emptyset$
5	$\emptyset$	$\{a - b\}$
6	$\{a - b\}$	$\emptyset$
7	$\{a - b\}$	$\emptyset$
8	$\emptyset$	—

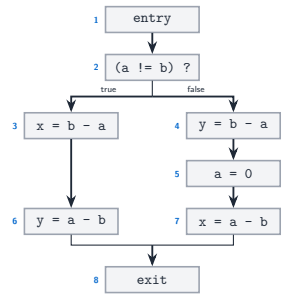
Saturated. This is the fixed point.



Very Busy: Pass 2 — Saturated

n	**IN**[n]	**OUT**[n]
1	—	$\{b - a\}$
2	$\{b - a\}$	$\{b - a\}$
3	$\{a - b, b - a\}$	$\{a - b\}$
4	$\{b - a\}$	$\emptyset$
5	$\emptyset$	$\{a - b\}$
6	$\{a - b\}$	$\emptyset$
7	$\{a - b\}$	$\emptyset$
8	$\emptyset$	—

Saturated. This is the fixed point.



Very Busy: the Result

- At **P** $= \mathrm{IN}[2]$: $\{\mathtt{b} - \mathtt{a}\}$

Very Busy: the Result

- At **P** $= \mathrm{IN}[2]$: $\{\mathtt{b} - \mathtt{a}\}$
- $\mathtt{b} - \mathtt{a}$ **is** very busy at P — matches our hand argument. ✓

Very Busy: the Result

- At **P** $= \mathrm{IN}[2]$: $\{\mathtt{b} - \mathtt{a}\}$
- $\mathtt{b} - \mathtt{a}$ **is** very busy at P — matches our hand argument. ✓
- $\mathtt{a} - \mathtt{b}$ **is not** — killed by $\mathtt{a} = 0$ on the false path. ✓

Very Busy: the Result

- At **P** $= \mathrm{IN}[2]$: $\{b - a\}$
- $b - a$ **is** very busy at P — matches our hand argument. ✓
- $a - b$ **is not** — killed by $a = 0$ on the false path. ✓
- $a - b$ *is* very busy at $\mathrm{IN}[6]$ and $\mathrm{IN}[7]$, just not before node 5.

Very Busy: the Result

- At **P** $= \mathrm{IN}[2]$: $\{b - a\}$
- $b - a$ **is** very busy at P — matches our hand argument. ✓
- $a - b$ **is not** — killed by $a = 0$ on the false path. ✓
- $a - b$ *is* very busy at $\mathrm{IN}[6]$ and $\mathrm{IN}[7]$, just not before node 5.

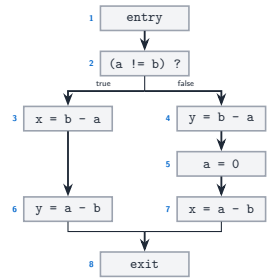
Quiz: Very Busy Expressions

Look at the graph. Is each expression very busy at the stated point?

1. b − a at the entry of node 3?

2. a − b at the entry of node 5?

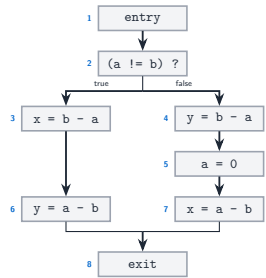
3. a − b at the entry of node 7?



Quiz: Very Busy Expressions

Look at the graph. Is each expression very busy at the stated point?

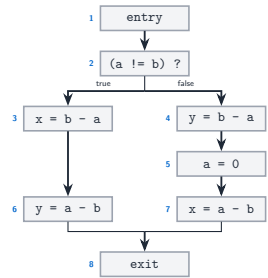
1. b − a at the entry of node 3?
   YES
2. a − b at the entry of node 5?
3. a − b at the entry of node 7?



Quiz: Very Busy Expressions

Look at the graph. Is each expression very busy at the stated point?

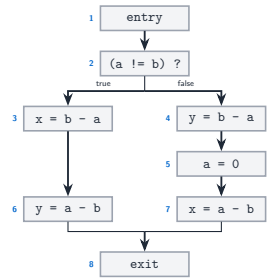
1. b − a at the entry of node 3?
   YES
2. a − b at the entry of node 5?
   NO
3. a − b at the entry of node 7?



Quiz: Very Busy Expressions

Look at the graph. Is each expression very busy at the stated point?

1. b − a at the entry of node 3?
   YES
2. a − b at the entry of node 5?
   NO
3. a − b at the entry of node 7?
   YES

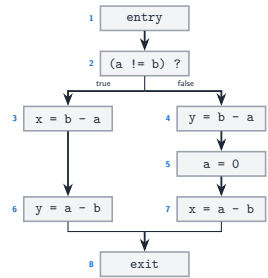


Quiz: Very Busy Expressions

Look at the graph. Is each expression very busy at the stated point?

1. b − a at the entry of node 3?
   YES
2. a − b at the entry of node 5?
   NO
3. a − b at the entry of node 7?
   YES

Table: they are $\mathrm{IN}[3]$, $\mathrm{IN}[5]$ and $\mathrm{IN}[7]$.



Application: Reducing Code Size

If an expression is very busy at a point, it *will* be computed on every path ahead. So compute it **once, here**, and reuse the result.

Application: Reducing Code Size

If an expression is very busy at a point, it *will* be computed on every path ahead. So compute it **once, here**, and reuse the result.

- `b − a` is very busy at P, and computed at *both* nodes 3 and 4.

Application: Reducing Code Size

If an expression is very busy at a point, it *will* be computed on every path ahead. So compute it **once, here**, and reuse the result.

- `b − a` is very busy at P, and computed at *both* nodes 3 and 4.
- Hoist it above the branch: compute `b − a` once at P, then reuse it.

Application: Reducing Code Size

If an expression is very busy at a point, it *will* be computed on every path ahead. So compute it **once, here**, and reuse the result.

- `b − a` is very busy at P, and computed at *both* nodes 3 and 4.
- Hoist it above the branch: compute `b − a` once at P, then reuse it.
- Two copies of the code become one. **The program shrinks.**

Application: Reducing Code Size

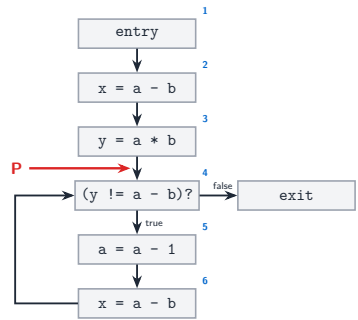
If an expression is very busy at a point, it *will* be computed on every path ahead. So compute it **once, here**, and reuse the result.

- `b − a` is very busy at P, and computed at *both* nodes 3 and 4.
- Hoist it above the branch: compute `b − a` once at P, then reuse it.
- Two copies of the code become one. **The program shrinks.**

Available Expression Analysis

What Makes an Expression Available?

An expression is **available** at a program point if, on *every* path reaching that point, it has already been computed and not since modified.

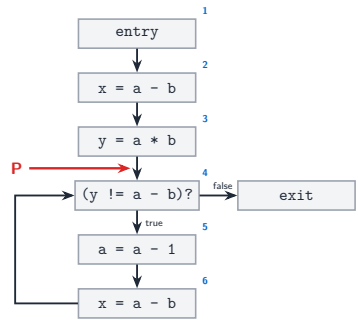


Available Expression Analysis

What Makes an Expression Available?

An expression is **available** at a program point if, on *every* path reaching that point, it has already been computed and not since modified.

Three expressions:
$a - b$, $a * b$, $a - 1$.



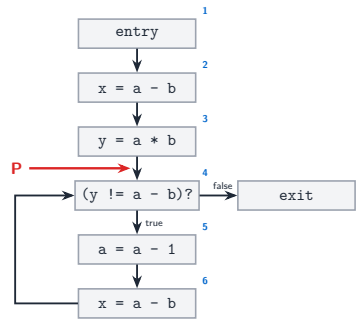
Available Expression Analysis

What Makes an Expression Available?

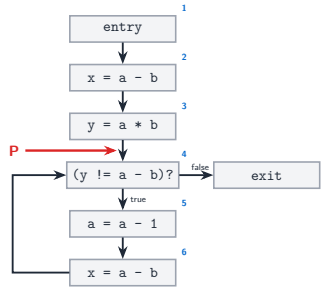
An expression is **available** at a program point if, on *every* path reaching that point, it has already been computed and not since modified.

Three expressions:
$a - b$, $a * b$, $a - 1$.

Consider **P**, the entry of node 4.



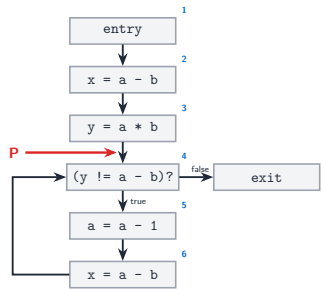
One Is Available, One Is Not



One Is Available, One Is Not

`a − b` is available at P.

Computed at node 2 on the way in;
computed at node 6 round the loop.
Both paths, nothing modified after.



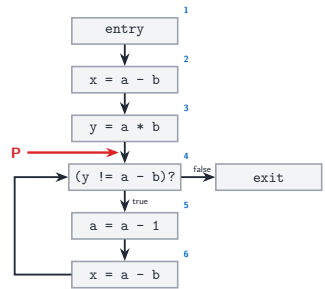
One Is Available, One Is Not

`a − b` is available at P.

Computed at node 2 on the way in;
computed at node 6 round the loop.
Both paths, nothing modified after.

`a * b` is *not* available at P.

Computed at node 3 on the direct path ✓
but round the loop node 5 sets $a = a - 1$
and it is never recomputed. ×



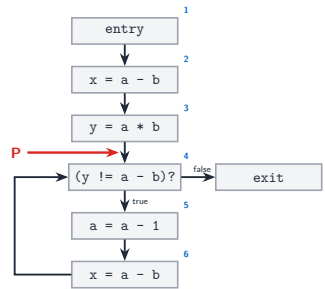
One Is Available, One Is Not

`a − b` is available at P.

Computed at node 2 on the way in;
computed at node 6 round the loop.
Both paths, nothing modified after.

`a * b` is *not* available at P.

Computed at node 3 on the direct path ✓
but round the loop node 5 sets $a = a - 1$
and it is never recomputed. ×



The Two Operations

1 — Merge at the entry (forward, so predecessors)

$$\mathrm{IN}[n] = \bigcap_{n' \in pred(n)} \mathrm{OUT}[n']$$

The Two Operations

1 — Merge at the entry (forward, so predecessors)

$$\mathrm{IN}[n] \;=\; \bigcap_{n' \in pred(n)} \mathrm{OUT}[n']$$

2 — Transfer through the node

$$\mathrm{OUT}[n] \;=\; \big(\mathrm{IN}[n] - \mathbf{KILL}[n]\big) \;\cup\; \mathbf{GEN}[n]$$

The Two Operations

1 — Merge at the entry (forward, so predecessors)

$$\mathrm{IN}[n] \;=\; \bigcap_{n' \in pred(n)} \mathrm{OUT}[n']$$

2 — Transfer through the node

$$\mathrm{OUT}[n] \;=\; \big(\mathrm{IN}[n] - \mathbf{KILL}[n]\big) \;\cup\; \mathbf{GEN}[n]$$

Same shape as reaching definitions — **but $\cap$ instead of $\cup$**.

The Two Operations

1 — Merge at the entry (forward, so predecessors)

$$\mathrm{IN}[n] \;=\; \bigcap_{n' \in pred(n)} \mathrm{OUT}[n']$$

2 — Transfer through the node

$$\mathrm{OUT}[n] \;=\; \big(\mathrm{IN}[n] - \mathbf{KILL}[n]\big) \;\cup\; \mathbf{GEN}[n]$$

Same shape as reaching definitions — **but $\cap$ instead of $\cup$**.

An expression is available at n only if it is available on *every* way of getting there.

Available: the GEN and KILL Rules

Statement at n	**GEN**$[n]$	**KILL**$[n]$
a condition		
an assignment $\mathtt{x} = \mathtt{a}$		

Available: the GEN and KILL Rules

Statement at n	**GEN**$[n]$	**KILL**$[n]$
a condition	$\emptyset$	$\emptyset$
an assignment $\mathtt{x} = \mathtt{a}$		

Available: the GEN and KILL Rules

Statement at n	**GEN**$[n]$	**KILL**$[n]$
a condition	$\emptyset$	$\emptyset$
an assignment $\texttt{x} = \texttt{a}$	$\{\texttt{a}\}$ if a does not use x	every expression using x

Available: the GEN and KILL Rules

Statement at n	GEN$[n]$	KILL$[n]$
a condition	$\emptyset$	$\emptyset$
an assignment $\mathtt{x} = \mathtt{a}$	$\{\mathtt{a}\}$ if a does not use x	every expression using x

Watch the side condition. $\mathtt{a} = \mathtt{a} - 1$ does *not* generate $\mathtt{a} - 1$: the assignment overwrites a in the same breath, so the computed value is stale at once.

Available: the GEN and KILL Rules

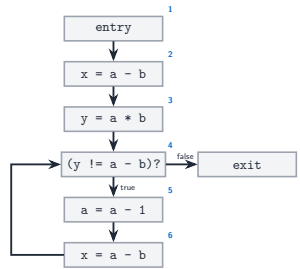
Statement at n	GEN$[n]$	KILL$[n]$
a condition	$\emptyset$	$\emptyset$
an assignment $\mathtt{x} = \mathtt{a}$	$\{\mathtt{a}\}$ if a does not use x	every expression using x

Watch the side condition. $\mathtt{a} = \mathtt{a} - 1$ does *not* generate $\mathtt{a} - 1$: the assignment overwrites a in the same breath, so the computed value is stale at once.

It still kills every expression using a — here, all three.

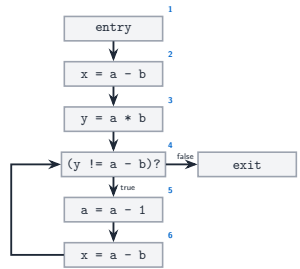
Available: GEN and KILL for the Program

n	Statement	GEN	KILL
2	$x = a - b$		
3	$y = a * b$		
4	$(y! = a - b)?$		
5	$a = a - 1$		
6	$x = a - b$		



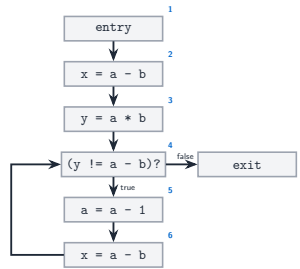
Available: GEN and KILL for the Program

n	Statement	GEN	KILL
2	$x = a - b$	$\{a - b\}$	$\emptyset$
3	$y = a * b$		
4	$(y! = a - b)?$		
5	$a = a - 1$		
6	$x = a - b$		



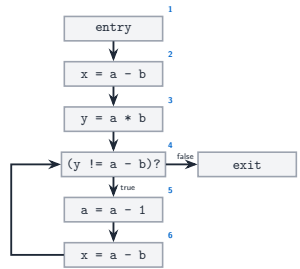
Available: GEN and KILL for the Program

n	Statement	GEN	KILL
2	$\mathtt{x} = \mathtt{a} - \mathtt{b}$	$\{\mathtt{a} - \mathtt{b}\}$	$\emptyset$
3	$\mathtt{y} = \mathtt{a} * \mathtt{b}$	$\{\mathtt{a} * \mathtt{b}\}$	$\emptyset$
4	$(\mathtt{y}! = \mathtt{a} - \mathtt{b})?$		
5	$\mathtt{a} = \mathtt{a} - 1$		
6	$\mathtt{x} = \mathtt{a} - \mathtt{b}$		



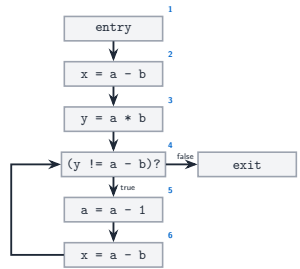
Available: GEN and KILL for the Program

n	Statement	GEN	KILL
2	$\mathtt{x} = \mathtt{a} - \mathtt{b}$	$\{\mathtt{a} - \mathtt{b}\}$	$\emptyset$
3	$\mathtt{y} = \mathtt{a} * \mathtt{b}$	$\{\mathtt{a} * \mathtt{b}\}$	$\emptyset$
4	$(\mathtt{y}! = \mathtt{a} - \mathtt{b})?$	$\emptyset$	$\emptyset$
5	$\mathtt{a} = \mathtt{a} - 1$		
6	$\mathtt{x} = \mathtt{a} - \mathtt{b}$		



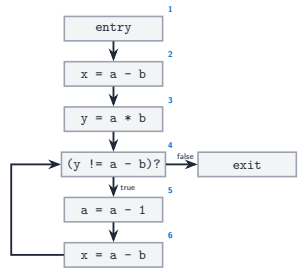
Available: GEN and KILL for the Program

n	Statement	GEN	KILL
2	$\mathtt{x} = \mathtt{a} - \mathtt{b}$	$\{\mathtt{a} - \mathtt{b}\}$	$\emptyset$
3	$\mathtt{y} = \mathtt{a} * \mathtt{b}$	$\{\mathtt{a} * \mathtt{b}\}$	$\emptyset$
4	$(\mathtt{y} != \mathtt{a} - \mathtt{b})?$	$\emptyset$	$\emptyset$
5	$\mathtt{a} = \mathtt{a} - 1$	$\emptyset$	$\{\mathtt{a} - \mathtt{b}, \mathtt{a} * \mathtt{b}, \mathtt{a} - 1\}$
6	$\mathtt{x} = \mathtt{a} - \mathtt{b}$		



Available: GEN and KILL for the Program

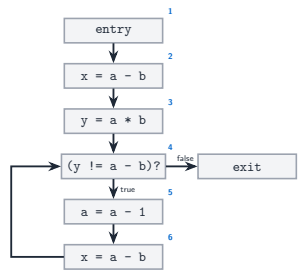
n	Statement	GEN	KILL
2	$\mathtt{x} = \mathtt{a} - \mathtt{b}$	$\{\mathtt{a} - \mathtt{b}\}$	$\emptyset$
3	$\mathtt{y} = \mathtt{a} * \mathtt{b}$	$\{\mathtt{a} * \mathtt{b}\}$	$\emptyset$
4	$(\mathtt{y} != \mathtt{a} - \mathtt{b})?$	$\emptyset$	$\emptyset$
5	$\mathtt{a} = \mathtt{a} - 1$	$\emptyset$	$\{\mathtt{a} - \mathtt{b}, \mathtt{a} * \mathtt{b}, \mathtt{a} - 1\}$
6	$\mathtt{x} = \mathtt{a} - \mathtt{b}$	$\{\mathtt{a} - \mathtt{b}\}$	$\emptyset$



Available: GEN and KILL for the Program

n	Statement	GEN	KILL
2	$\mathtt{x} = \mathtt{a} - \mathtt{b}$	$\{\mathtt{a} - \mathtt{b}\}$	$\emptyset$
3	$\mathtt{y} = \mathtt{a} * \mathtt{b}$	$\{\mathtt{a} * \mathtt{b}\}$	$\emptyset$
4	$(\mathtt{y} != \mathtt{a} - \mathtt{b})?$	$\emptyset$	$\emptyset$
5	$\mathtt{a} = \mathtt{a} - 1$	$\emptyset$	$\{\mathtt{a} - \mathtt{b}, \mathtt{a} * \mathtt{b}, \mathtt{a} - 1\}$
6	$\mathtt{x} = \mathtt{a} - \mathtt{b}$	$\{\mathtt{a} - \mathtt{b}\}$	$\emptyset$

$\mathtt{a} - 1$ is killed at node 5 and generated *nowhere* — so it is never available at any point.

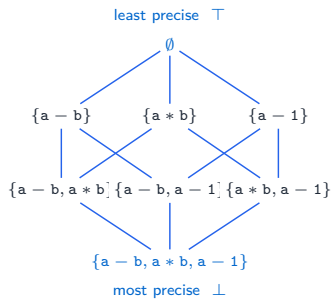


Available: the Abstract Domain

- Three expressions $\Rightarrow 2^3 = 8$ abstract values.

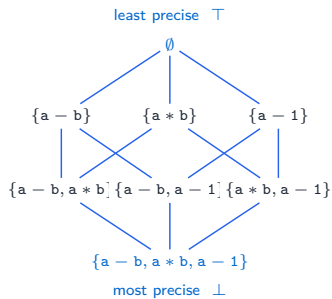
Available: the Abstract Domain

- Three expressions $\Rightarrow 2^3 = 8$ abstract values.
- Ordered by **reverse** set inclusion, as for very busy expressions.



Available: the Abstract Domain

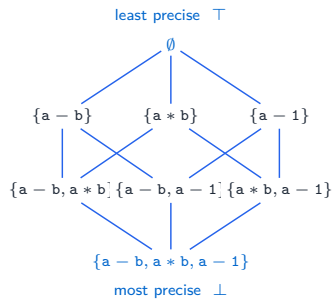
- Three expressions $\Rightarrow 2^3 = 8$ abstract values.
- Ordered by **reverse** set inclusion, as for very busy expressions.
- $\bot$ = all expressions; $\top = \emptyset$.



Available: the Abstract Domain

- Three expressions $\Rightarrow 2^3 = 8$ abstract values.
- Ordered by **reverse** set inclusion, as for very busy expressions.
- $\bot$ = all expressions; $\top = \emptyset$.

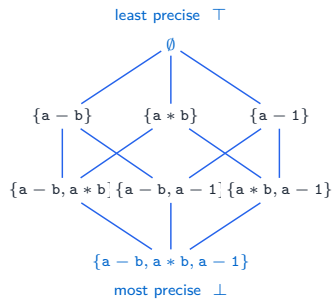
Sets start *full* and **shrink**. Boundary: $\mathrm{OUT}[\mathit{entry}] = \emptyset$.



Available: the Abstract Domain

- Three expressions $\Rightarrow 2^3 = 8$ abstract values.
- Ordered by **reverse** set inclusion, as for very busy expressions.
- $\bot$ = all expressions; $\top = \emptyset$.

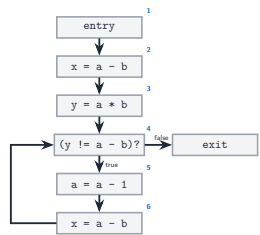
Sets start *full* and **shrink**. Boundary: $\mathrm{OUT}[\mathit{entry}] = \emptyset$.



Available: Pass 1 — Forward

Init: all $\{a - b, a * b, a - 1\}$, except $\mathrm{OUT}[1] = \emptyset$. Visiting $2 \ldots 7$.

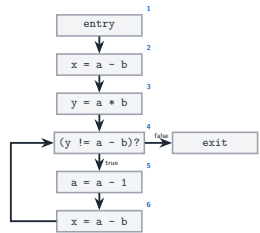
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\emptyset$
2	$\{a - b, a * b, a - 1\}$	$\{a - b, a * b, a - 1\}$
3	$\{a - b, a * b, a - 1\}$	$\{a - b, a * b, a - 1\}$
4	$\{a - b, a * b, a - 1\}$	$\{a - b, a * b, a - 1\}$
5	$\{a - b, a * b, a - 1\}$	$\{a - b, a * b, a - 1\}$
6	$\{a - b, a * b, a - 1\}$	$\{a - b, a * b, a - 1\}$
7	$\{a - b, a * b, a - 1\}$	—



Available: Pass 1 — Forward

Init: all $\{a - b, a * b, a - 1\}$, except $\mathrm{OUT}[1] = \emptyset$. Visiting $2 \ldots 7$.

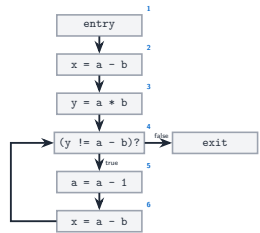
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\emptyset$
2	$\emptyset$	$\{a - b\}$
3	$\{a - b, a * b, a - 1\}$	$\{a - b, a * b, a - 1\}$
4	$\{a - b, a * b, a - 1\}$	$\{a - b, a * b, a - 1\}$
5	$\{a - b, a * b, a - 1\}$	$\{a - b, a * b, a - 1\}$
6	$\{a - b, a * b, a - 1\}$	$\{a - b, a * b, a - 1\}$
7	$\{a - b, a * b, a - 1\}$	—



Available: Pass 1 — Forward

Init: all $\{a - b, a * b, a - 1\}$, except $\mathrm{OUT}[1] = \emptyset$. Visiting $2 \ldots 7$.

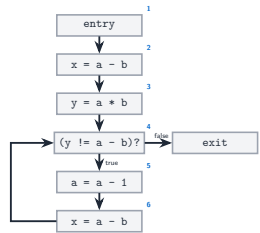
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\emptyset$
2	$\emptyset$	$\{a - b\}$
3	$\{a - b\}$	$\{a - b, a * b\}$
4	$\{a - b, a * b, a - 1\}$	$\{a - b, a * b, a - 1\}$
5	$\{a - b, a * b, a - 1\}$	$\{a - b, a * b, a - 1\}$
6	$\{a - b, a * b, a - 1\}$	$\{a - b, a * b, a - 1\}$
7	$\{a - b, a * b, a - 1\}$	—



Available: Pass 1 — Forward

Init: all $\{a - b, a * b, a - 1\}$, except $\mathrm{OUT}[1] = \emptyset$. Visiting $2 \ldots 7$.

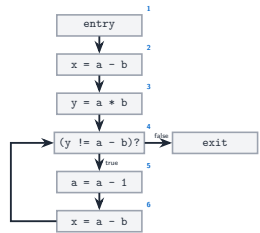
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\emptyset$
2	$\emptyset$	$\{a - b\}$
3	$\{a - b\}$	$\{a - b, a * b\}$
4	$\{a - b, a * b\}$	$\{a - b, a * b\}$
5	$\{a - b, a * b, a - 1\}$	$\{a - b, a * b, a - 1\}$
6	$\{a - b, a * b, a - 1\}$	$\{a - b, a * b, a - 1\}$
7	$\{a - b, a * b, a - 1\}$	—



Available: Pass 1 — Forward

Init: all $\{a - b, a * b, a - 1\}$, except $\mathrm{OUT}[1] = \emptyset$. Visiting $2 \ldots 7$.

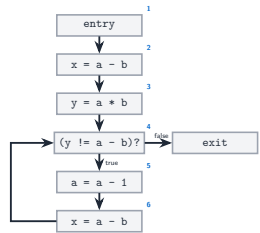
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\emptyset$
2	$\emptyset$	$\{a - b\}$
3	$\{a - b\}$	$\{a - b, a * b\}$
4	$\{a - b, a * b\}$	$\{a - b, a * b\}$
5	$\{a - b, a * b\}$	$\emptyset$
6	$\{a - b, a * b, a - 1\}$	$\{a - b, a * b, a - 1\}$
7	$\{a - b, a * b, a - 1\}$	—



Available: Pass 1 — Forward

Init: all $\{a - b, a * b, a - 1\}$, except $\mathrm{OUT}[1] = \emptyset$. Visiting $2 \ldots 7$.

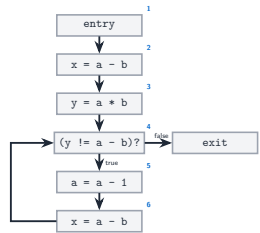
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\emptyset$
2	$\emptyset$	$\{a - b\}$
3	$\{a - b\}$	$\{a - b, a * b\}$
4	$\{a - b, a * b\}$	$\{a - b, a * b\}$
5	$\{a - b, a * b\}$	$\emptyset$
6	$\emptyset$	$\{a - b\}$
7	$\{a - b, a * b, a - 1\}$	—



Available: Pass 1 — Forward

Init: all $\{a - b, a * b, a - 1\}$, except $\mathrm{OUT}[1] = \emptyset$. Visiting $2 \ldots 7$.

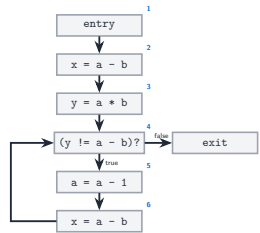
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\emptyset$
2	$\emptyset$	$\{a - b\}$
3	$\{a - b\}$	$\{a - b, a * b\}$
4	$\{a - b, a * b\}$	$\{a - b, a * b\}$
5	$\{a - b, a * b\}$	$\emptyset$
6	$\emptyset$	$\{a - b\}$
7	$\{a - b, a * b\}$	—



Available: Pass 1 — Forward

Init: all $\{a - b, a * b, a - 1\}$, except $\mathrm{OUT}[1] = \emptyset$. Visiting $2 \ldots 7$.

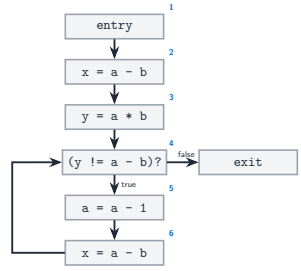
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\emptyset$
2	$\emptyset$	$\{a - b\}$
3	$\{a - b\}$	$\{a - b, a * b\}$
4	$\{a - b, a * b\}$	$\{a - b, a * b\}$
5	$\{a - b, a * b\}$	$\emptyset$
6	$\emptyset$	$\{a - b\}$
7	$\{a - b, a * b\}$	—



At node 4 we intersected with $\mathrm{OUT}[6]$, which was still the full set — so this answer is not final.

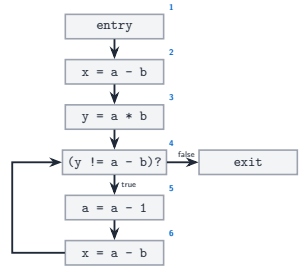
Available: Pass 2

n	IN[n]	OUT[n]
1	—	$\emptyset$
2	$\emptyset$	$\{a - b\}$
3	$\{a - b\}$	$\{a - b, a * b\}$
4	$\{a - b, a * b\}$	$\{a - b, a * b\}$
5	$\{a - b, a * b\}$	$\emptyset$
6	$\emptyset$	$\{a - b\}$
7	$\{a - b, a * b\}$	—



Available: Pass 2

n	IN[n]	OUT[n]
1	—	$\emptyset$
2	$\emptyset$	$\{a - b\}$
3	$\{a - b\}$	$\{a - b, a * b\}$
4	$\{a - b\}$	$\{a - b\}$
5	$\{a - b, a * b\}$	$\emptyset$
6	$\emptyset$	$\{a - b\}$
7	$\{a - b, a * b\}$	—

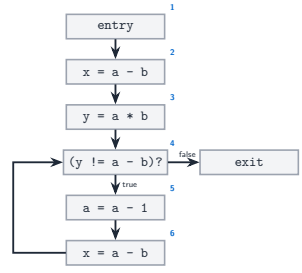


$$\text{IN}[4] = \text{OUT}[3] \cap \text{OUT}[6] =$$

$$\{a - b, a * b\} \cap \{a - b\}$$

Available: Pass 2

n	IN[n]	OUT[n]
1	—	$\emptyset$
2	$\emptyset$	$\{a - b\}$
3	$\{a - b\}$	$\{a - b, a * b\}$
4	$\{a - b\}$	$\{a - b\}$
5	$\{a - b\}$	$\emptyset$
6	$\emptyset$	$\{a - b\}$
7	$\{a - b, a * b\}$	—

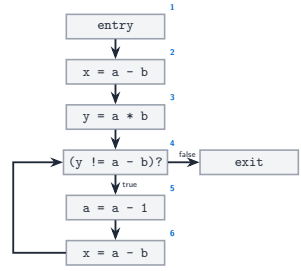


$$\text{IN}[4] = \text{OUT}[3] \cap \text{OUT}[6] =$$

$$\{a - b, a * b\} \cap \{a - b\}$$

Available: Pass 2

n	IN[n]	OUT[n]
1	—	$\emptyset$
2	$\emptyset$	$\{a - b\}$
3	$\{a - b\}$	$\{a - b, a * b\}$
4	$\{a - b\}$	$\{a - b\}$
5	$\{a - b\}$	$\emptyset$
6	$\emptyset$	$\{a - b\}$
7	$\{a - b\}$	—



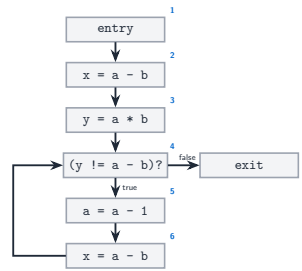
$$\text{IN}[4] = \text{OUT}[3] \cap \text{OUT}[6] =$$

$$\{a - b, a * b\} \cap \{a - b\}$$

Available: Pass 2

n	IN[n]	OUT[n]
1	—	$\emptyset$
2	$\emptyset$	$\{a - b\}$
3	$\{a - b\}$	$\{a - b, a * b\}$
4	$\{a - b\}$	$\{a - b\}$
5	$\{a - b\}$	$\emptyset$
6	$\emptyset$	$\{a - b\}$
7	$\{a - b\}$	—

Pass 3 changes nothing — this is the fixed point.



$$\text{IN}[4] = \text{OUT}[3] \cap \text{OUT}[6] =$$

$$\{a - b, a * b\} \cap \{a - b\}$$

Available: the Result

- At **P** $= \mathrm{IN}[4]$: $\{\mathtt{a} - \mathtt{b}\}$

Available: the Result

- At **P** $= \mathrm{IN}[4]$: $\{\mathtt{a} - \mathtt{b}\}$
- $\mathtt{a} - \mathtt{b}$ **is** available — matches the hand argument. ✓

Available: the Result

- At **P** $= \mathrm{IN}[4]$: $\{\mathtt{a} - \mathtt{b}\}$
- $\mathtt{a} - \mathtt{b}$ **is** available — matches the hand argument. ✓
- $\mathtt{a} * \mathtt{b}$ **is not** — killed round the loop. ✓

Available: the Result

- At **P** $= \mathrm{IN}[4]$: $\{\mathtt{a} - \mathtt{b}\}$
- $\mathtt{a} - \mathtt{b}$ **is** available — matches the hand argument. ✓
- $\mathtt{a} * \mathtt{b}$ **is not** — killed round the loop. ✓
- $\mathtt{a} - 1$ is available **nowhere** — generated by no node.

Available: the Result

- At **P** $= \mathrm{IN}[4]$: $\{\mathtt{a} - \mathtt{b}\}$
- $\mathtt{a} - \mathtt{b}$ **is** available — matches the hand argument. ✓
- $\mathtt{a} * \mathtt{b}$ **is not** — killed round the loop. ✓
- $\mathtt{a} - 1$ is available **nowhere** — generated by no node.

$\mathtt{a} - \mathtt{b}$ is available at nodes 4, 5 and 7 — everywhere the loop condition can be reached.

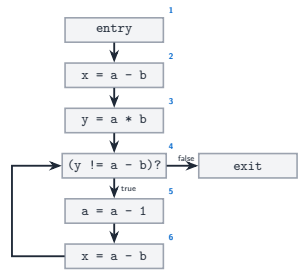
Quiz: Available Expressions

Which expressions are available at the stated points?

1. a − b at the entry of node 3?

2. a ∗ b at the entry of node 5?

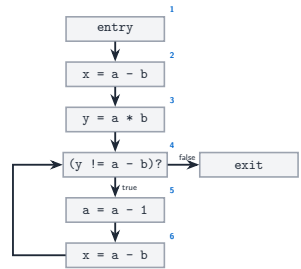
3. Anything at all at the entry of node 6?



Quiz: Available Expressions

Which expressions are available at the stated points?

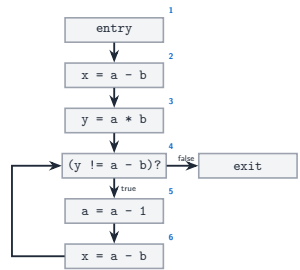
1. a − b at the entry of node 3?
   YES
2. a ∗ b at the entry of node 5?
3. Anything at all at the entry of node 6?



Quiz: Available Expressions

Which expressions are available at the stated points?

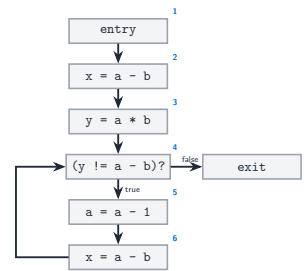
1. a − b at the entry of node 3?
   YES
2. a ∗ b at the entry of node 5?
   NO
3. Anything at all at the entry of node 6?



Quiz: Available Expressions

Which expressions are available at the stated points?

1. a − b at the entry of node 3?
   YES
2. a ∗ b at the entry of node 5?
   NO
3. Anything at all at the entry of node 6? **NO**

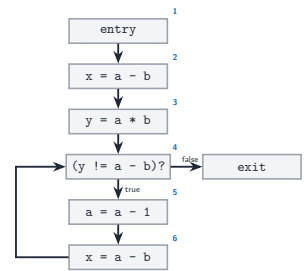


Quiz: Available Expressions

Which expressions are available at the stated points?

1. a − b at the entry of node 3?
   YES
2. a ∗ b at the entry of node 5?
   NO
3. Anything at all at the entry of node 6? **NO**

a − 1 is available *nowhere* — no node ever generates it.



Application: Avoiding Recomputation

If an expression is available at a point, its value has already been computed on every path here.

So do not compute it again — **reuse** it.

Application: Avoiding Recomputation

If an expression is available at a point, its value has already been computed on every path here. So do not compute it again — **reuse** it.

- $\mathtt{a} - \mathtt{b}$ is available at node 4, which tests $\mathtt{y} != \mathtt{a} - \mathtt{b}$.

Application: Avoiding Recomputation

If an expression is available at a point, its value has already been computed on every path here.

So do not compute it again — **reuse** it.

- $\mathtt{a} - \mathtt{b}$ is available at node 4, which tests $\mathtt{y} != \mathtt{a} - \mathtt{b}$.
- That test can reuse the value computed at node 2 or node 6 instead of recomputing it.

Application: Avoiding Recomputation

If an expression is available at a point, its value has already been computed on every path here.

So do not compute it again — **reuse** it.

- $\mathtt{a} - \mathtt{b}$ is available at node 4, which tests $\mathtt{y} != \mathtt{a} - \mathtt{b}$.
- That test can reuse the value computed at node 2 or node 6 instead of recomputing it.
- **Common subexpression elimination.** The program gets *faster*.

Application: Avoiding Recomputation

If an expression is available at a point, its value has already been computed on every path here.

So do not compute it again — **reuse** it.

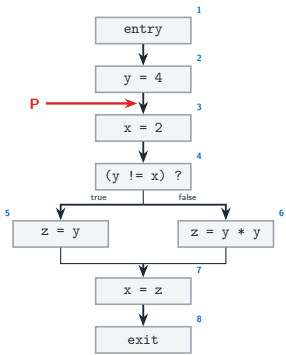
- $\mathtt{a} - \mathtt{b}$ is available at node 4, which tests $\mathtt{y} != \mathtt{a} - \mathtt{b}$.
- That test can reuse the value computed at node 2 or node 6 instead of recomputing it.
- **Common subexpression elimination.** The program gets *faster*.

Contrast very busy expressions, which saved code *size*. This one saves *time*.

Live Variable Analysis

What Makes a Variable Live?

A variable is **live** at a program point if there is a path from that point to a *use* of the variable which does not redefine it first.

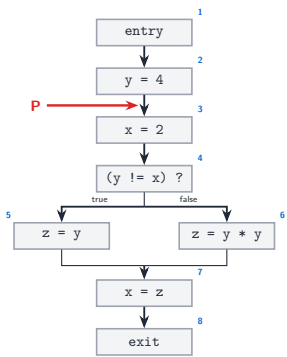


Live Variable Analysis

What Makes a Variable Live?

A variable is **live** at a program point if there is a path from that point to a *use* of the variable which does not redefine it first.

Three variables: `x`, `y`, `z`.



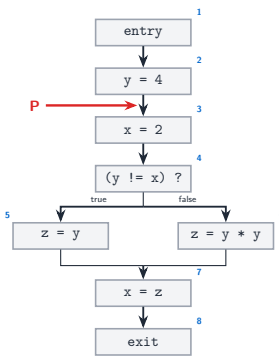
Live Variable Analysis

What Makes a Variable Live?

A variable is **live** at a program point if there is a path from that point to a *use* of the variable which does not redefine it first.

Three variables: `x`, `y`, `z`.

Consider **P**, the entry of node 3.



Live Variable Analysis

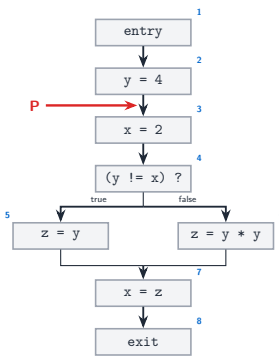
What Makes a Variable Live?

A variable is **live** at a program point if there is a path from that point to a *use* of the variable which does not redefine it first.

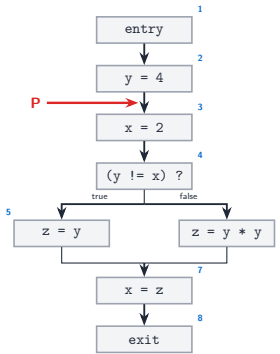
Three variables: `x`, `y`, `z`.

Consider **P**, the entry of node 3.

"Live" means *still needed*. A dead variable's value can be thrown away.

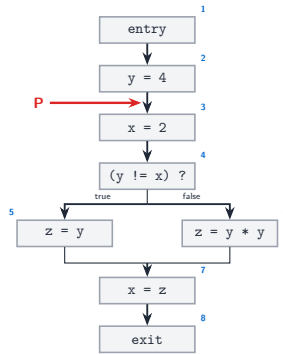


Only One Variable Is Live at P



Only One Variable Is Live at P

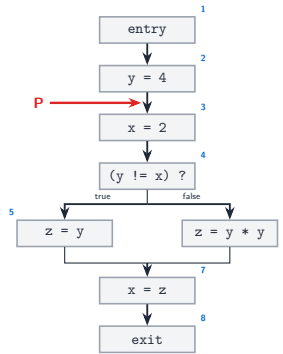
y is live at P. used by the test at node 4, and not redefined on the way.



Only One Variable Is Live at P

y is live at P. used by the test at node 4, and not redefined on the way.

x is not live at P. it *is* used at node 4 — but node 3 redefines it first.

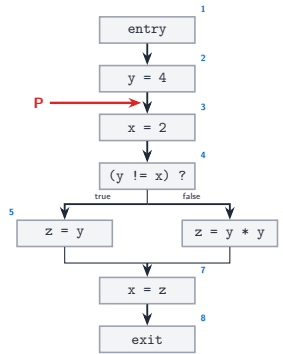


Only One Variable Is Live at P

`y` is live at P. used by the test at node 4, and not redefined on the way.

`x` is not live at P. it *is* used at node 4 — but node 3 redefines it first.

`z` is not live at P. used at node 7, but nodes 5 and 6 both redefine it first.

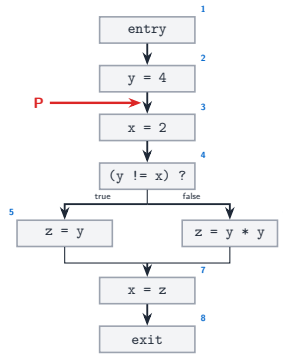


Only One Variable Is Live at P

y is live at P. used by the test at node 4, and not redefined on the way.

x is not live at P. it *is* used at node 4 — but node 3 redefines it first.

z is not live at P. used at node 7, but nodes 5 and 6 both redefine it first.



For z, *every* path redefines it first — so even a *may* analysis rules it out.

The Two Operations, Reversed

1 — Merge at the exit (backward, so successors)

$$\mathrm{OUT}[n] = \bigcup_{n' \in succ(n)} \mathrm{IN}[n']$$

The Two Operations, Reversed

1 — Merge at the exit (backward, so successors)

$$\mathrm{OUT}[n] \;=\; \bigcup_{n' \in succ(n)} \mathrm{IN}[n']$$

2 — Transfer through the node

$$\mathrm{IN}[n] \;=\; \big(\mathrm{OUT}[n] - \mathbf{KILL}[n]\big) \;\cup\; \mathbf{GEN}[n]$$

The Two Operations, Reversed

1 — Merge at the exit (backward, so successors)

$$\mathrm{OUT}[n] \;=\; \bigcup_{n' \in succ(n)} \mathrm{IN}[n']$$

2 — Transfer through the node

$$\mathrm{IN}[n] \;=\; \big(\mathrm{OUT}[n] - \mathbf{KILL}[n]\big) \;\cup\; \mathbf{GEN}[n]$$

GEN$[n]$ — variables **used** at n. **KILL**$[n]$ — variables **defined** at n.

The Two Operations, Reversed

1 — Merge at the exit (backward, so successors)

$$\mathrm{OUT}[n] \;=\; \bigcup_{n' \in succ(n)} \mathrm{IN}[n']$$

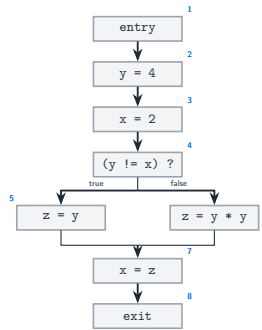
2 — Transfer through the node

$$\mathrm{IN}[n] \;=\; \big(\mathrm{OUT}[n] - \mathbf{KILL}[n]\big) \;\cup\; \mathbf{GEN}[n]$$

GEN$[n]$ — variables **used** at n. **KILL**$[n]$ — variables **defined** at n.

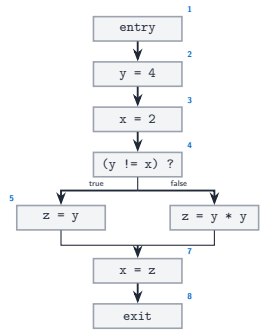
Live: GEN and KILL — Reads and Writes

n	Statement	GEN (used)	KILL (defined)
2	$y = 4$		
3	$x = 2$		
4	$(y! = x)?$		
5	$z = y$		
6	$z = y * y$		
7	$x = z$		



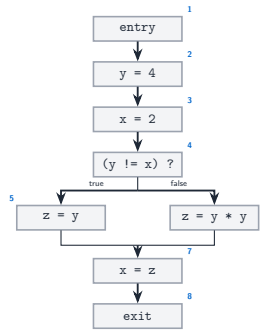
Live: GEN and KILL — Reads and Writes

n	Statement	GEN (used)	KILL (defined)
2	$y = 4$	$\emptyset$	$\{y\}$
3	$x = 2$		
4	$(y! = x)?$		
5	$z = y$		
6	$z = y * y$		
7	$x = z$		



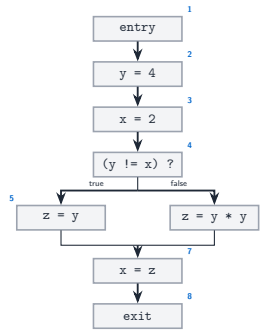
Live: GEN and KILL — Reads and Writes

n	Statement	GEN (used)	KILL (defined)
2	$y = 4$	$\emptyset$	$\{y\}$
3	$x = 2$	$\emptyset$	$\{x\}$
4	$(y! = x)?$		
5	$z = y$		
6	$z = y * y$		
7	$x = z$		



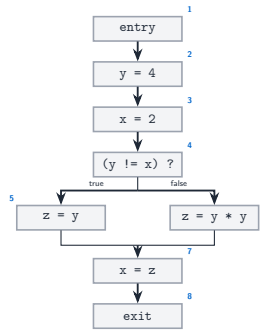
Live: GEN and KILL — Reads and Writes

n	Statement	GEN (used)	KILL (defined)
2	$y = 4$	$\emptyset$	$\{y\}$
3	$x = 2$	$\emptyset$	$\{x\}$
4	$(y! = x)?$	$\{x, y\}$	$\emptyset$
5	$z = y$		
6	$z = y * y$		
7	$x = z$		



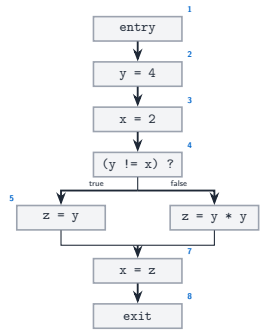
Live: GEN and KILL — Reads and Writes

n	Statement	GEN (used)	KILL (defined)
2	$y = 4$	$\emptyset$	$\{y\}$
3	$x = 2$	$\emptyset$	$\{x\}$
4	$(y! = x)?$	$\{x, y\}$	$\emptyset$
5	$z = y$	$\{y\}$	$\{z\}$
6	$z = y * y$		
7	$x = z$		



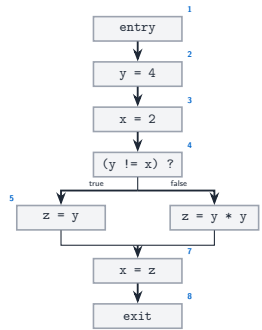
Live: GEN and KILL — Reads and Writes

n	Statement	GEN (used)	KILL (defined)
2	$y = 4$	$\emptyset$	$\{y\}$
3	$x = 2$	$\emptyset$	$\{x\}$
4	$(y! = x)?$	$\{x, y\}$	$\emptyset$
5	$z = y$	$\{y\}$	$\{z\}$
6	$z = y * y$	$\{y\}$	$\{z\}$
7	$x = z$		



Live: GEN and KILL — Reads and Writes

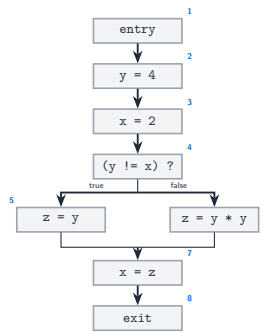
n	Statement	GEN (used)	KILL (defined)
2	$y = 4$	$\emptyset$	$\{y\}$
3	$x = 2$	$\emptyset$	$\{x\}$
4	$(y! = x)?$	$\{x, y\}$	$\emptyset$
5	$z = y$	$\{y\}$	$\{z\}$
6	$z = y * y$	$\{y\}$	$\{z\}$
7	$x = z$	$\{z\}$	$\{x\}$



Live: GEN and KILL — Reads and Writes

n	Statement	GEN (used)	KILL (defined)
2	$y = 4$	$\emptyset$	$\{y\}$
3	$x = 2$	$\emptyset$	$\{x\}$
4	$(y! = x)?$	$\{x, y\}$	$\emptyset$
5	$z = y$	$\{y\}$	$\{z\}$
6	$z = y * y$	$\{y\}$	$\{z\}$
7	$x = z$	$\{z\}$	$\{x\}$

A **condition** generates but never kills — it reads and writes nothing.

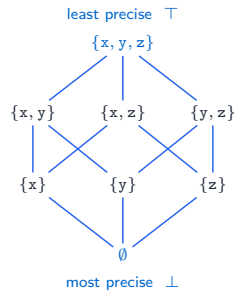


Live: the Abstract Domain

- Three variables $\Rightarrow 2^3 = 8$ abstract values.

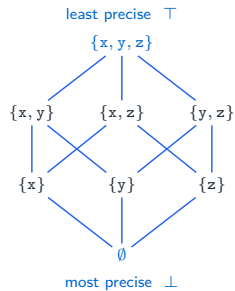
Live: the Abstract Domain

- Three variables $\Rightarrow 2^3 = 8$ abstract values.
- Ordered by **plain** set inclusion — as for reaching definitions.



Live: the Abstract Domain

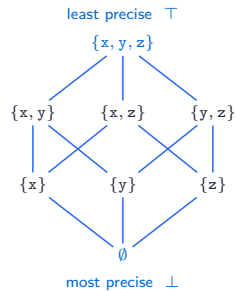
- Three variables $\Rightarrow 2^3 = 8$ abstract values.
- Ordered by **plain** set inclusion — as for reaching definitions.
- $\bot = \emptyset$; $\top = \{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$.



Live: the Abstract Domain

- Three variables $\Rightarrow 2^3 = 8$ abstract values.
- Ordered by **plain** set inclusion — as for reaching definitions.
- $\bot = \emptyset$; $\top = \{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$.

Sets start *empty* and **grow**. Boundary: $\mathrm{IN}[\mathit{exit}] = \emptyset$.

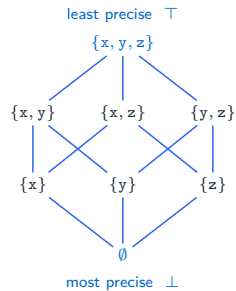


Live: the Abstract Domain

- Three variables $\Rightarrow 2^3 = 8$ abstract values.
- Ordered by **plain** set inclusion — as for reaching definitions.
- $\bot = \emptyset$; $\top = \{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$.

Sets start *empty* and **grow**. Boundary: $\mathrm{IN}[\mathit{exit}] = \emptyset$.

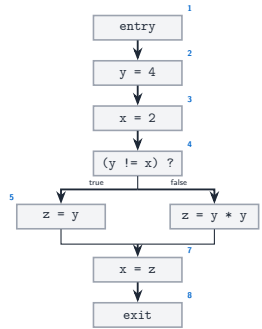
A *smaller* set of live variables is a stronger claim, so smaller is more precise.



Live: Pass 1 — Working Backwards

Init: all $\emptyset$. Visiting $7, 6, 5, 4, 3, 2, 1$.

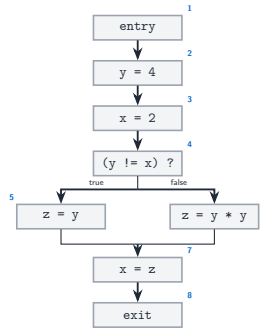
n	**IN**[n]	**OUT**[n]
1	—	$\emptyset$
2	$\emptyset$	$\emptyset$
3	$\emptyset$	$\emptyset$
4	$\emptyset$	$\emptyset$
5	$\emptyset$	$\emptyset$
6	$\emptyset$	$\emptyset$
7	{z}	$\emptyset$
8	$\emptyset$	—



Live: Pass 1 — Working Backwards

Init: all $\emptyset$. Visiting $7, 6, 5, 4, 3, 2, 1$.

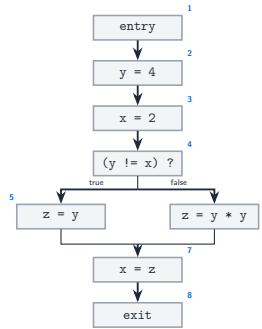
n	**IN**[n]	**OUT**[n]
1	—	$\emptyset$
2	$\emptyset$	$\emptyset$
3	$\emptyset$	$\emptyset$
4	$\emptyset$	$\emptyset$
5	$\emptyset$	$\emptyset$
6	{y}	{z}
7	{z}	$\emptyset$
8	$\emptyset$	—



Live: Pass 1 — Working Backwards

Init: all $\emptyset$. Visiting $7, 6, 5, 4, 3, 2, 1$.

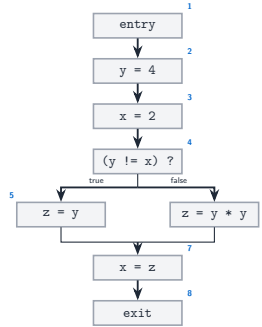
n	**IN**[n]	**OUT**[n]
1	—	$\emptyset$
2	$\emptyset$	$\emptyset$
3	$\emptyset$	$\emptyset$
4	$\emptyset$	$\emptyset$
5	$\{y\}$	$\{z\}$
6	$\{y\}$	$\{z\}$
7	$\{z\}$	$\emptyset$
8	$\emptyset$	—



Live: Pass 1 — Working Backwards

Init: all $\emptyset$. Visiting $7, 6, 5, 4, 3, 2, 1$.

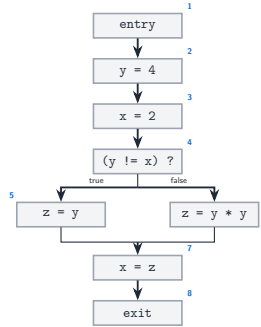
n	**IN**[n]	**OUT**[n]
1	—	$\emptyset$
2	$\emptyset$	$\emptyset$
3	$\emptyset$	$\emptyset$
4	$\{x, y\}$	$\{y\}$
5	$\{y\}$	$\{z\}$
6	$\{y\}$	$\{z\}$
7	$\{z\}$	$\emptyset$
8	$\emptyset$	—



Live: Pass 1 — Working Backwards

Init: all $\emptyset$. Visiting $7, 6, 5, 4, 3, 2, 1$.

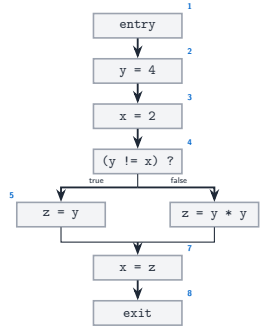
n	**IN**$[n]$	**OUT**$[n]$
1	—	$\emptyset$
2	$\emptyset$	$\emptyset$
3	$\{\texttt{y}\}$	$\{\texttt{x}, \texttt{y}\}$
4	$\{\texttt{x}, \texttt{y}\}$	$\{\texttt{y}\}$
5	$\{\texttt{y}\}$	$\{\texttt{z}\}$
6	$\{\texttt{y}\}$	$\{\texttt{z}\}$
7	$\{\texttt{z}\}$	$\emptyset$
8	$\emptyset$	—



Live: Pass 1 — Working Backwards

Init: all $\emptyset$. Visiting $7, 6, 5, 4, 3, 2, 1$.

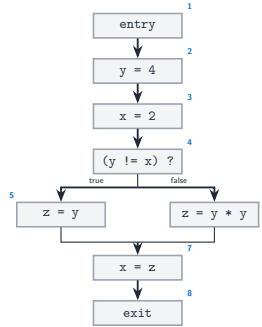
n	**IN**[n]	**OUT**[n]
1	—	$\emptyset$
2	$\emptyset$	$\{y\}$
3	$\{y\}$	$\{x, y\}$
4	$\{x, y\}$	$\{y\}$
5	$\{y\}$	$\{z\}$
6	$\{y\}$	$\{z\}$
7	$\{z\}$	$\emptyset$
8	$\emptyset$	—



Live: Pass 1 — Working Backwards

Init: all $\emptyset$. Visiting $7, 6, 5, 4, 3, 2, 1$.

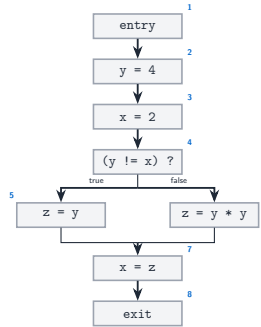
n	**IN**[n]	**OUT**[n]
1	—	$\emptyset$
2	$\emptyset$	$\{y\}$
3	$\{y\}$	$\{x, y\}$
4	$\{x, y\}$	$\{y\}$
5	$\{y\}$	$\{z\}$
6	$\{y\}$	$\{z\}$
7	$\{z\}$	$\emptyset$
8	$\emptyset$	—



Node 4 is where the two branches rejoin — and both contribute y.

Live: Pass 2 — Saturated

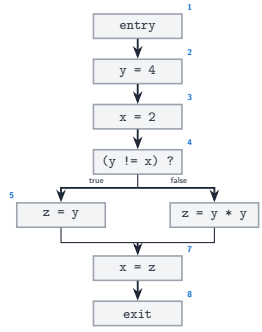
n	**IN**[n]	**OUT**[n]
1	—	$\emptyset$
2	$\emptyset$	$\{y\}$
3	$\{y\}$	$\{x, y\}$
4	$\{x, y\}$	$\{y\}$
5	$\{y\}$	$\{z\}$
6	$\{y\}$	$\{z\}$
7	$\{z\}$	$\emptyset$
8	$\emptyset$	—



Live: Pass 2 — Saturated

n	**IN**[n]	**OUT**[n]
1	—	$\emptyset$
2	$\emptyset$	$\{\mathtt{y}\}$
3	$\{\mathtt{y}\}$	$\{\mathtt{x}, \mathtt{y}\}$
4	$\{\mathtt{x}, \mathtt{y}\}$	$\{\mathtt{y}\}$
5	$\{\mathtt{y}\}$	$\{\mathtt{z}\}$
6	$\{\mathtt{y}\}$	$\{\mathtt{z}\}$
7	$\{\mathtt{z}\}$	$\emptyset$
8	$\emptyset$	—

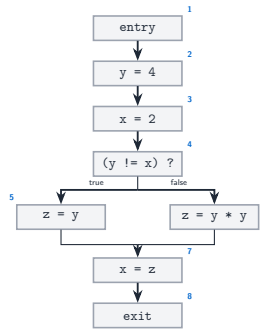
Saturated. This is the fixed point.



Live: Pass 2 — Saturated

n	**IN**[n]	**OUT**[n]
1	—	$\emptyset$
2	$\emptyset$	$\{y\}$
3	$\{y\}$	$\{x, y\}$
4	$\{x, y\}$	$\{y\}$
5	$\{y\}$	$\{z\}$
6	$\{y\}$	$\{z\}$
7	$\{z\}$	$\emptyset$
8	$\emptyset$	—

Saturated. This is the fixed point.

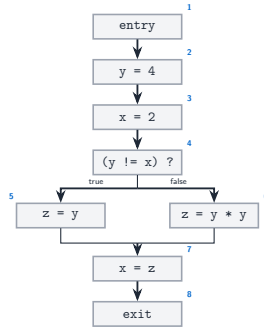


At no point are more than two variables live at once.

Quiz: Live Variables

Which variables are live at the stated points?

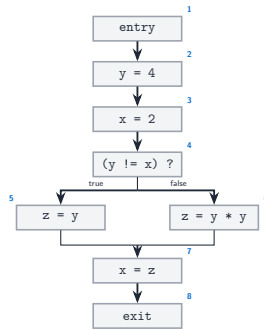
1. x at the entry of node 4?
2. z at the entry of node 5?
3. y at the exit of node 7?



Quiz: Live Variables

Which variables are live at the stated points?

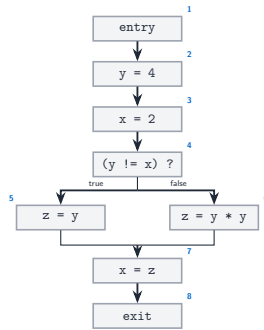
1. x at the entry of node 4? **YES**
2. z at the entry of node 5?
3. y at the exit of node 7?



Quiz: Live Variables

Which variables are live at the stated points?

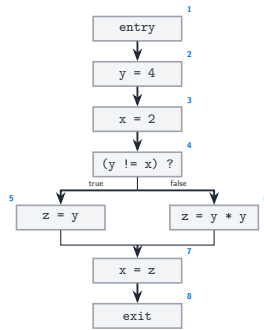
1. x at the entry of node 4? **YES**
2. z at the entry of node 5? **NO**
3. y at the exit of node 7?



Quiz: Live Variables

Which variables are live at the stated points?

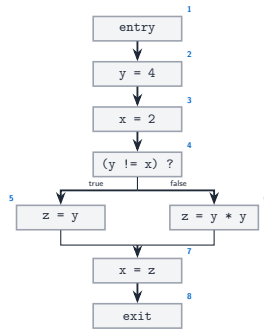
1. x at the entry of node 4? **YES**
2. z at the entry of node 5? **NO**
3. y at the exit of node 7? **NO**



Quiz: Live Variables

Which variables are live at the stated points?

1. x at the entry of node 4? **YES**
2. z at the entry of node 5? **NO**
3. y at the exit of node 7? **NO**



Nothing is live at the exit — the program is over, so no value is needed.

Application: Register Allocation

Two variables that are never live at the same time can share a register — neither value is needed while the other is in use.

Application: Register Allocation

Two variables that are never live at the same time can share a register — neither value is needed while the other is in use.

- The program has **three** variables `x`, `y`, `z`.

Application: Register Allocation

Two variables that are never live at the same time can share a register — neither value is needed while the other is in use.

- The program has **three** variables `x`, `y`, `z`.
- But at most **two** are live at any point.

Application: Register Allocation

Two variables that are never live at the same time can share a register — neither value is needed while the other is in use.

- The program has **three** variables `x`, `y`, `z`.
- But at most **two** are live at any point.
- So the generated code needs only **two registers**, not three.

Application: Register Allocation

Two variables that are never live at the same time can share a register — neither value is needed while the other is in use.

- The program has **three** variables `x`, `y`, `z`.
- But at most **two** are live at any point.
- So the generated code needs only **two registers**, not three.

Fewer registers means less spilling to memory — and register allocation is the compiler stage that most affects the speed of the generated code.

May and Must: the Classification

	may **(some path)**	***must*** **(every path)**
forward		
backward		

May and Must: the Classification

	may **(some path)**	***must*** **(every path)**
forward	Reaching definitions *uninitialised variables*	
backward		

May and Must: the Classification

	may **(some path)**	***must*** **(every path)**
forward	Reaching definitions *uninitialised variables*	
backward		Very busy expressions *reduce code size*

May and Must: the Classification

	may **(some path)**	***must*** **(every path)**
forward	Reaching definitions *uninitialised variables*	Available expressions *avoid recomputation*
backward		Very busy expressions *reduce code size*

May and Must: the Classification

	may **(some path)**	***must*** **(every path)**
forward	Reaching definitions *uninitialised variables*	Available expressions *avoid recomputation*
backward	Live variables *register allocation*	Very busy expressions *reduce code size*

May and Must: the Classification

	may **(some path)**	***must*** **(every path)**
forward	Reaching definitions *uninitialised variables*	Available expressions *avoid recomputation*
backward	Live variables *register allocation*	Very busy expressions *reduce code size*

May and Must: the Classification

	may **(some path)**	***must*** **(every path)**
forward	Reaching definitions *uninitialised variables*	Available expressions *avoid recomputation*
backward	Live variables *register allocation*	Very busy expressions *reduce code size*

Summary

Analysis	Dir.	Kind	Merge	$\perp$	Application
Reaching definitions	fwd	may	$\cup$	$\emptyset$	uninitialised variables
Very busy expressions	bwd	must	$\cap$	all	reduce code size
Available expressions	fwd	must	$\cap$	all	avoid recomputation
Live variables	**bwd**	**may**	$\cup$	$\emptyset$	**register allocation**