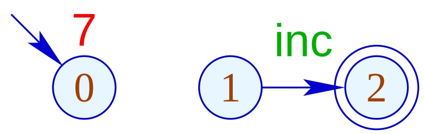
Caveat:

- Reachability of all program points cannot be abandoned! Consider:



where $\mathbb{D} = \mathbb{N} \cup \{\infty\}$

Then:

$$\mathcal{I}[2] = \mathsf{inc}\,0 = 1$$
$$\mathcal{I}^*[2] = \bigsqcup \emptyset = 0$$

- Unreachable program points can always be thrown away :-)

Summary and Application:

→ The effects of edges of the analysis of availability of expressions are distributive:

$$\begin{array}{lll} (a \cup (x_1 \cap x_2))\backslash b & = & ((a \cup x_1) \cap (a \cup x_2))\backslash b \\ & = & ((a \cup x_1)\backslash b) \cap ((a \cup x_2)\backslash b) \end{array}$$

Summary and Application:

→ The effects of edges of the analysis of availability of expressions are distributive:

$$\begin{aligned}(a \cup (x_1 \cap x_2))\backslash b &= ((a \cup x_1) \cap (a \cup x_2))\backslash b \\ &= ((a \cup x_1)\backslash b) \cap ((a \cup x_2)\backslash b)\end{aligned}$$

→ If all effects of edges are distributive, then the MOP can be computed by means of the constraint system and RR-iteration. :-)

Summary and Application:

→ The effects of edges of the analysis of availability of expressions are distributive:

$$
\begin{aligned}
(a \cup (x_1 \cap x_2)) \backslash b &= ((a \cup x_1) \cap (a \cup x_2)) \backslash b \\
&= ((a \cup x_1) \backslash b) \cap ((a \cup x_2) \backslash b)
\end{aligned}
$$

→ If all effects of edges are distributive, then the MOP can be computed by means of the constraint system and RR-iteration. :-)

→ If not all effects of edges are distributive, then RR-iteration for the constraint system at least returns a safe upper bound to the MOP :-)

1.2 Removing Assignments to Dead Variables

Example:

$$\begin{array}{ll} 1: & x = y + 2; \\ 2: & y = 5; \\ 3: & x = y + 3; \end{array}$$

The value of x at program points 1, 2 is over-written before it can be used.

Therefore, we call the variable x dead at these program points :-)

Note:

- → Assignments to dead variables can be removed ;-)
- → Such inefficiencies may originate from other transformations.

Note:

$\rightarrow$ Assignments to dead variables can be removed ;-)

$\rightarrow$ Such inefficiencies may originate from other transformations.

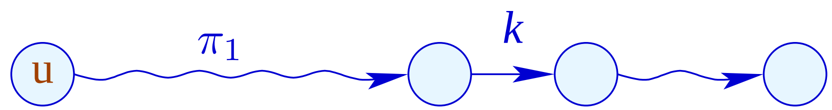
Formal Definition:

The variable x is called live at u along the path π starting at u relative to a set X of variables either:

if $x \in X$ and π does not contain a definition of x; or:

if π can be decomposed into: $\pi = \pi_1 \, k \, \pi_2$ such that:

- k is a use of x ; and
- π_1 does not contain a definition of x.

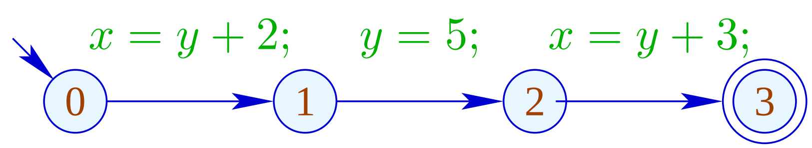


Thereby, the set of all defined or used variables at an edge $k = (\_, lab, \_)$ is defined by:

lab	used	defined
;	$\emptyset$	$\emptyset$
$\text{Pos}\,(e)$	$Vars\,(e)$	$\emptyset$
$\text{Neg}\,(e)$	$Vars\,(e)$	$\emptyset$
$x = e;$	$Vars\,(e)$	$\{x\}$
$x = M[e];$	$Vars\,(e)$	$\{x\}$
$M[e_1] = e_2;$	$Vars\,(e_1) \cup Vars\,(e_2)$	$\emptyset$

A variable x which is not live at u along π (relative to X) is called dead at u along π (relative to X).

Example:



where $X = \emptyset$. Then we observe:

	live	dead
0	$\{y\}$	$\{x\}$
1	$\emptyset$	$\{x, y\}$
2	$\{y\}$	$\{x\}$
3	$\emptyset$	$\{x, y\}$

The variable x is live at u (relative to X) if x is live at u along some path to the exit (relative to X). Otherwise, x is called dead at u (relative to X).

The variable x is live at u (relative to X) if x is live at u along some path to the exit (relative to X). Otherwise, x is called dead at u (relative to X).

Question:

How can the sets of all dead/live variables be computed for every u ???

The variable x is live at u (relative to X) if x is live at u along some path to the exit (relative to X). Otherwise, x is called dead at u (relative to X).

Question:

How can the sets of all dead/live variables be computed for every u ???

Idea:

For every edge $k = (u, \_, v)$, define a function $[\![k]\!]^\sharp$ which transforms the set of variables which are live at v into the set of variables which are live at u ...

Let $\mathbb{L} = 2^{Vars}$.

For $k = (\_, lab, \_)$, define $[\![k]\!]^\sharp = [\![lab]\!]^\sharp$ by:

$$
\begin{array}{lll}
[\![;]\!]^\sharp\, L & = & L \\
[\![\text{Pos}(e)]\!]^\sharp\, L & = & [\![\text{Neg}(e)]\!]^\sharp\, L \quad = \quad L \cup Vars(e) \\
[\![x = e;]\!]^\sharp\, L & = & (L \backslash \{x\}) \cup Vars(e) \\
[\![x = M[e];]\!]^\sharp\, L & = & (L \backslash \{x\}) \cup Vars(e) \\
[\![M[e_1] = e_2;]\!]^\sharp\, L & = & L \cup Vars(e_1) \cup Vars(e_2)
\end{array}
$$

Let $\mathbb{L} = 2^{Vars}$.

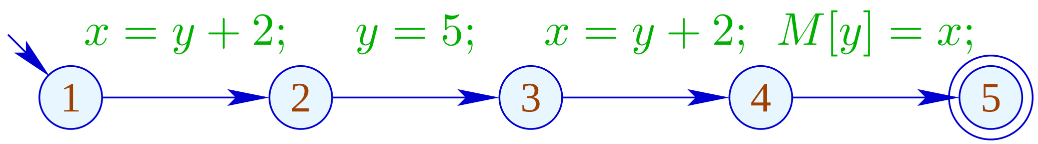
For $k = (\_, lab, \_)$, define $[\![k]\!]^\sharp = [\![lab]\!]^\sharp$ by:

$$
\begin{array}{lll}
[\![;]\!]^\sharp\, L & = & L \\
[\![\text{Pos}(e)]\!]^\sharp\, L & = & [\![\text{Neg}(e)]\!]^\sharp\, L \;=\; L \cup Vars(e) \\
[\![x = e;]\!]^\sharp\, L & = & (L \backslash \{x\}) \cup Vars(e) \\
[\![x = M[e];]\!]^\sharp\, L & = & (L \backslash \{x\}) \cup Vars(e) \\
[\![M[e_1] = e_2;]\!]^\sharp\, L & = & L \cup Vars(e_1) \cup Vars(e_2)
\end{array}
$$

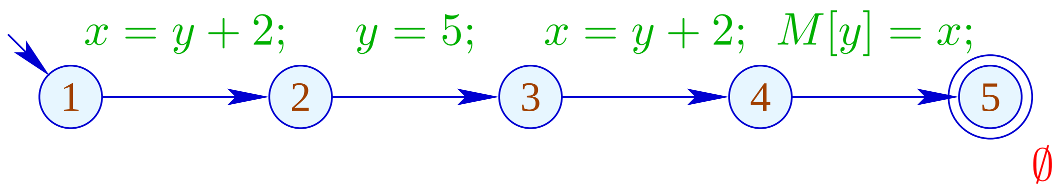
$[\![k]\!]^\sharp$ can again be composed to the effects of $[\![\pi]\!]^\sharp$ of paths $\pi = k_1 \ldots k_r$ by:

$$[\![\pi]\!]^\sharp = [\![k_1]\!]^\sharp \circ \ldots \circ [\![k_r]\!]^\sharp$$

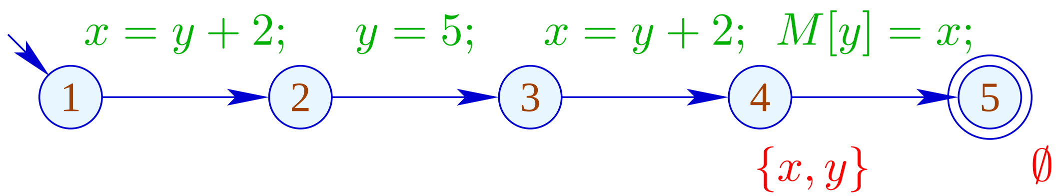
We verify that these definitions are meaningful :-)



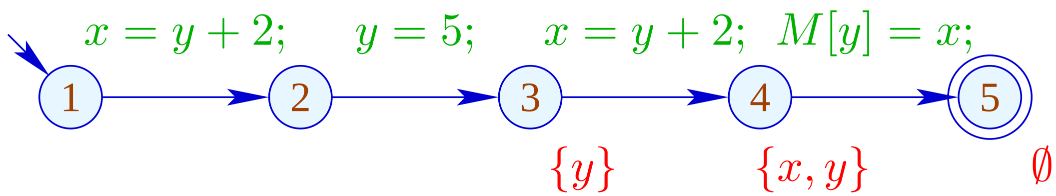
We verify that these definitions are meaningful :-)



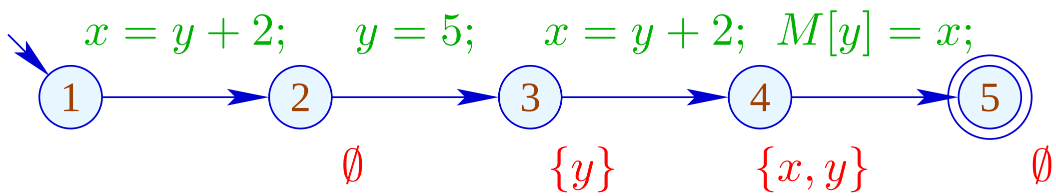
We verify that these definitions are meaningful :-)



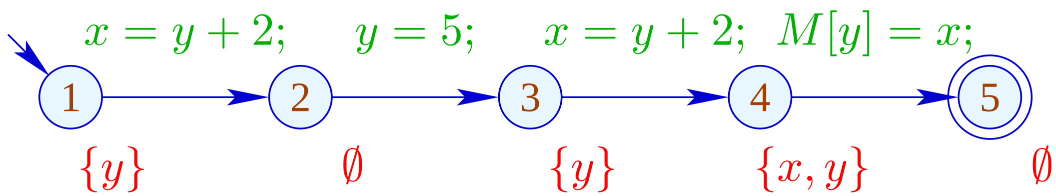
We verify that these definitions are meaningful :-)



We verify that these definitions are meaningful :-)



We verify that these definitions are meaningful :-)



The set of variables which are live at u then is given by:

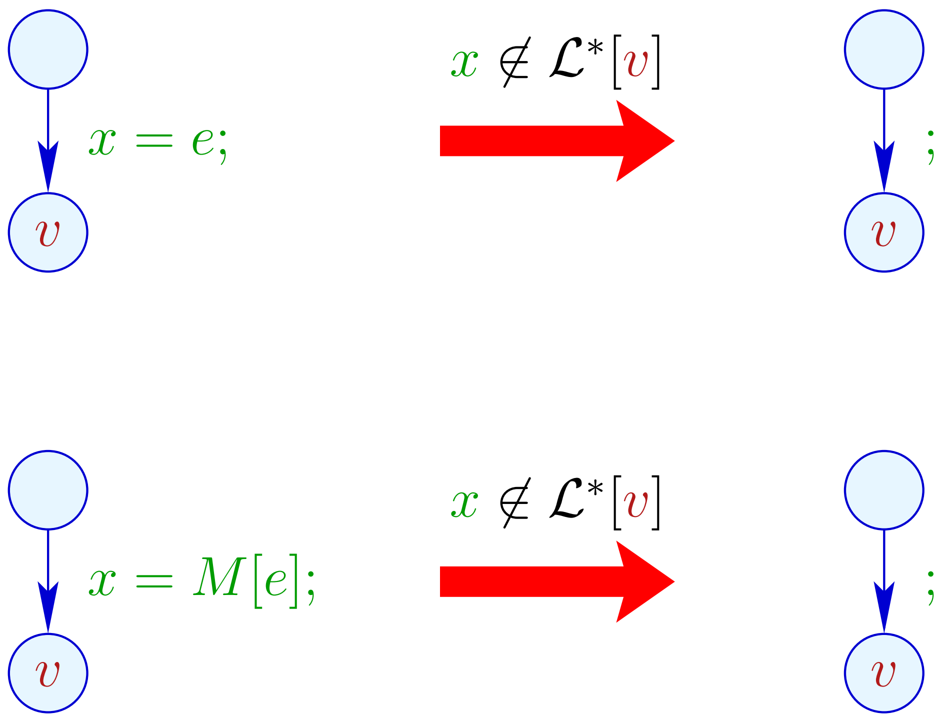
$$\mathcal{L}^*[u] \;=\; \bigcup\{[\![\pi]\!]^\sharp\, X \mid \pi : u \to^* \mathit{stop}\}$$

... literally:

- The paths start in u :-)

  $\Longrightarrow$ As partial ordering for $\mathbb{L}$ we use $\sqsubseteq \,=\, \subseteq$.
- The set of variables which are live at program exit is given by the set X :-)

Transformation 2:



Correctness Proof:

- → Correctness of the effects of edges: If L is the set of variables which are live at the exit of the path π , then $[\![\pi]\!]^\sharp\, L$ is the set of variables which are live at the beginning of π :-)
- → Correctness of the transformation along a path: If the value of a variable is accessed, this variable is necessarily live. The value of dead variables thus is irrelevant :-)
- → Correctness of the transformation: In any execution of the transformed programs, the live variables always receive the same values :-))

Computation of the sets $\mathcal{L}^*[u]$:

(1) Collecting constraints:

$$\begin{array}{lll} \mathcal{L}[stop] & \supseteq & X \\ \mathcal{L}[u] & \supseteq & [\![k]\!]^\sharp\,(\mathcal{L}[v]) \qquad k = (u, \_, v) \quad \text{edge} \end{array}$$

(2) Solving the constraint system by means of RR iteration.

Since $\mathbb{L}$ is finite, the iteration will terminate :-)

(3) If the exit is (formally) reachable from every program point, then the smallest solution $\mathcal{L}$ of the constraint system equals $\mathcal{L}^*$ since all $[\![k]\!]^\sharp$ are distributive :-))

Computation of the sets $\mathcal{L}^*[u]$:

(1) Collecting constraints:

$$\mathcal{L}[stop] \supseteq X$$
$$\mathcal{L}[u] \supseteq [\![k]\!]^\sharp (\mathcal{L}[v]) \qquad k = (u, \_, v) \quad \text{edge}$$

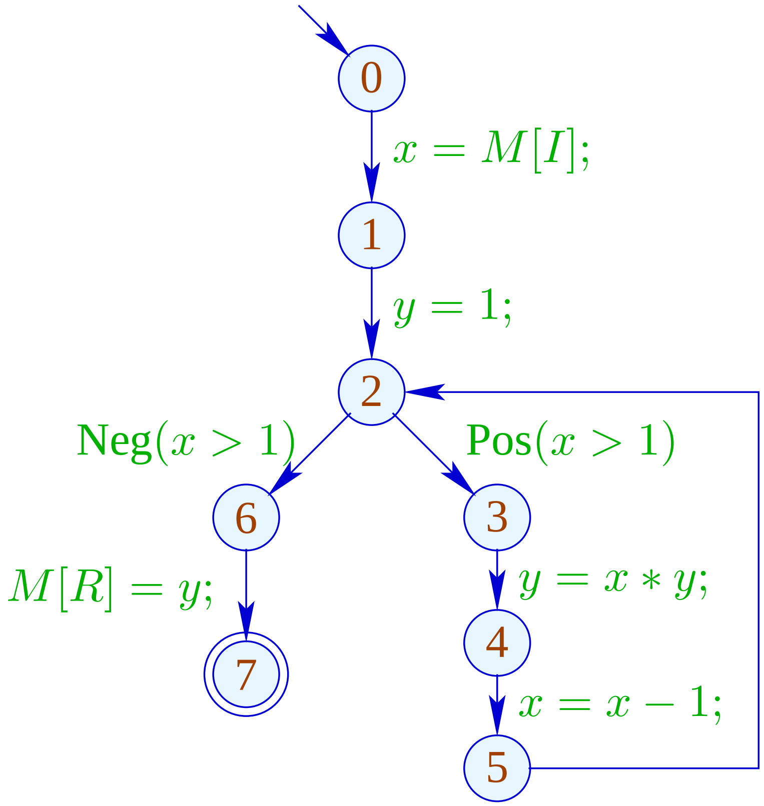
(2) Solving the constraint system by means of RR iteration.

Since $\mathbb{L}$ is finite, the iteration will terminate :-)

(3) If the exit is (formally) reachable from every program point, then the smallest solution $\mathcal{L}$ of the constraint system equals $\mathcal{L}^*$ since all $[\![k]\!]^\sharp$ are distributive :-))

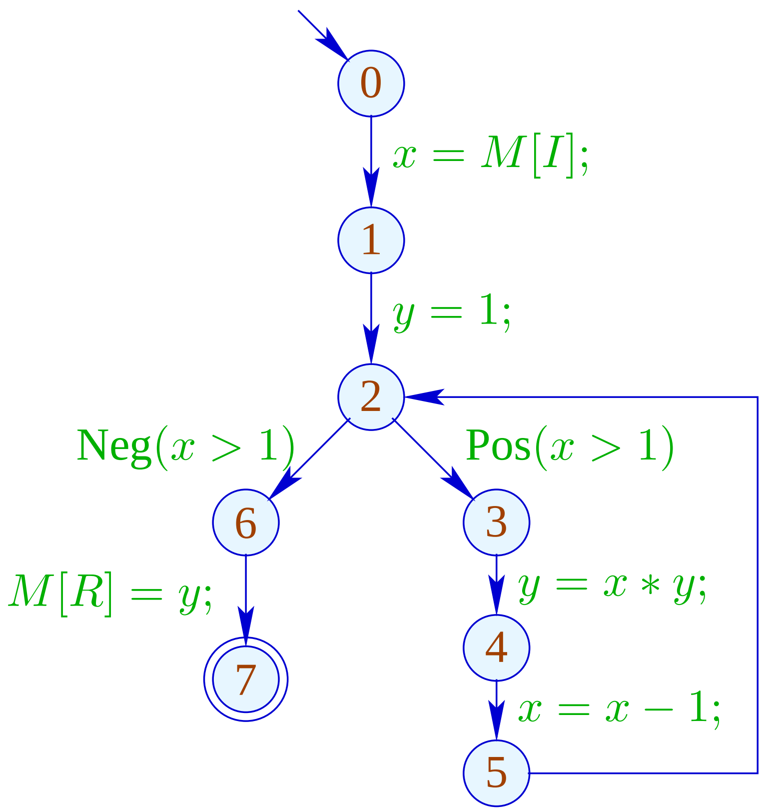
Caveat: The information is propagated backwards !!!

Example:



$$
\begin{array}{lcl}
\mathcal{L}[0] & \supseteq & (\mathcal{L}[1] \backslash \{x\}) \cup \{I\} \\
\mathcal{L}[1] & \supseteq & \mathcal{L}[2] \backslash \{y\} \\
\mathcal{L}[2] & \supseteq & (\mathcal{L}[6] \cup \{x\}) \cup (\mathcal{L}[3] \cup \{x\}) \\
\mathcal{L}[3] & \supseteq & (\mathcal{L}[4] \backslash \{y\}) \cup \{x, y\} \\
\mathcal{L}[4] & \supseteq & (\mathcal{L}[5] \backslash \{x\}) \cup \{x\} \\
\mathcal{L}[5] & \supseteq & \mathcal{L}[2] \\
\mathcal{L}[6] & \supseteq & \mathcal{L}[7] \cup \{y, R\} \\
\mathcal{L}[7] & \supseteq & \emptyset
\end{array}
$$

Example:

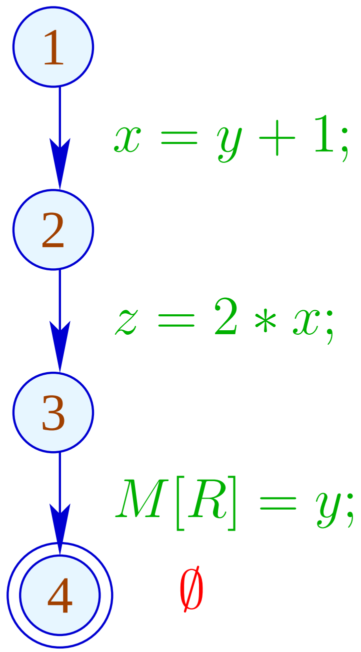


	1	2
7	$\emptyset$	
6	$\{y, R\}$	
2	$\{x, y, R\}$	dito
5	$\{x, y, R\}$	
4	$\{x, y, R\}$	
3	$\{x, y, R\}$	
1	$\{x, R\}$	
0	$\{I, R\}$	

The left-hand side of no assignment is dead :-)

Caveat:

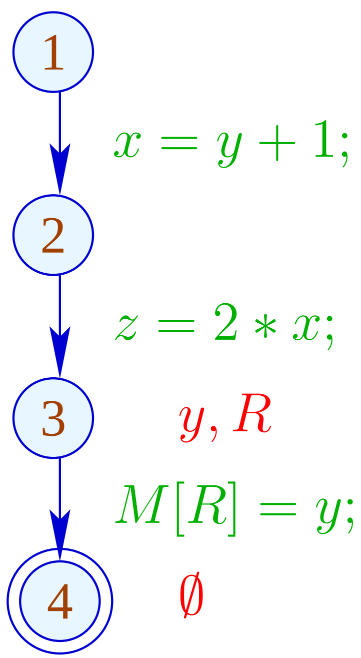
Removal of assignments to dead variables may kill further variables:



The left-hand side of no assignment is dead :-)

Caveat:

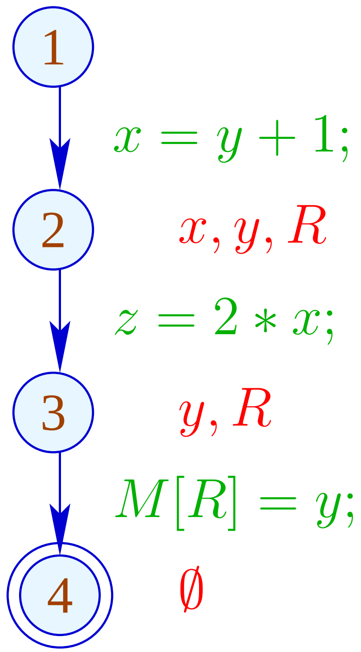
Removal of assignments to dead variables may kill further variables:



The left-hand side of no assignment is dead :-)

Caveat:

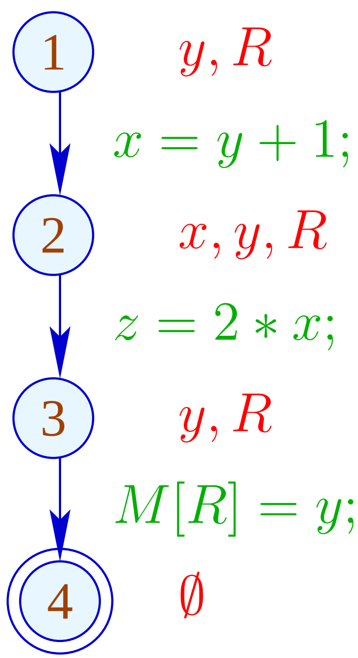
Removal of assignments to dead variables may kill further variables:



The left-hand side of no assignment is dead :-)

Caveat:

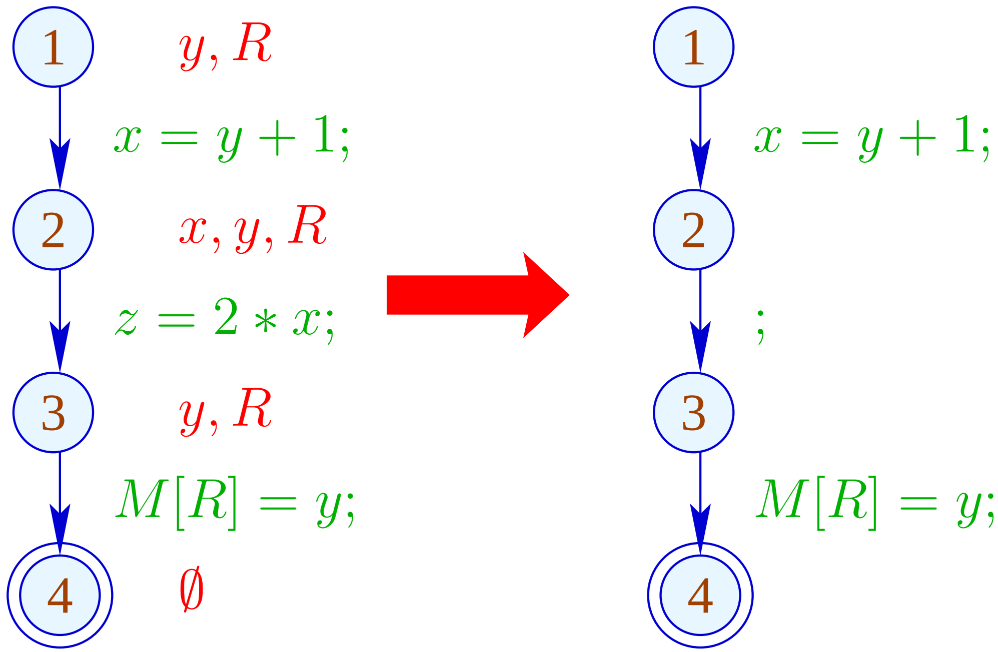
Removal of assignments to dead variables may kill further variables:



The left-hand side of no assignment is dead :-)

Caveat:

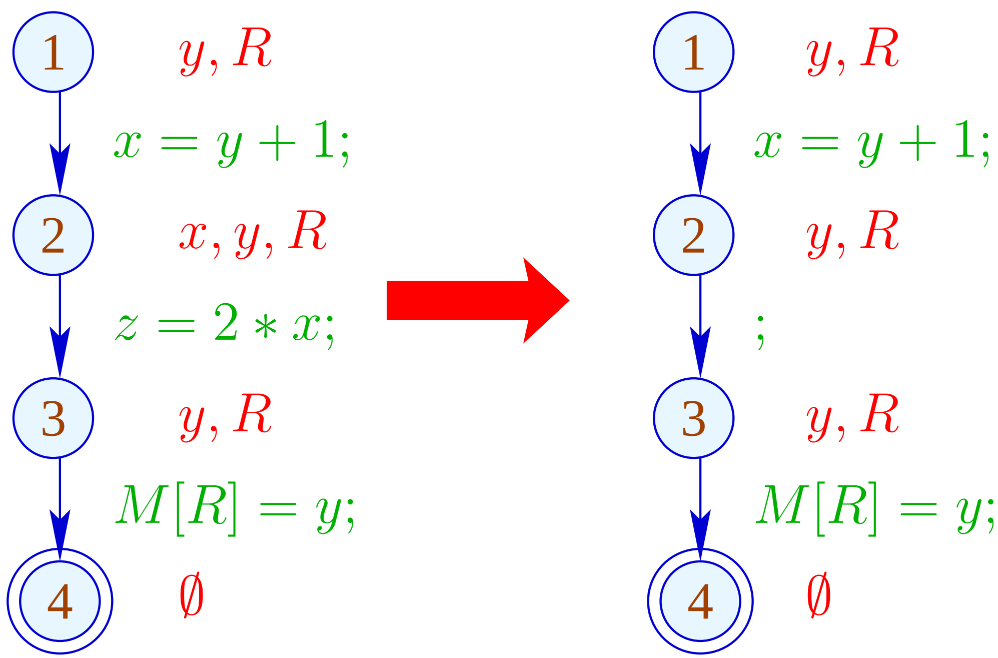
Removal of assignments to dead variables may kill further variables:



The left-hand side of no assignment is dead :-)

Caveat:

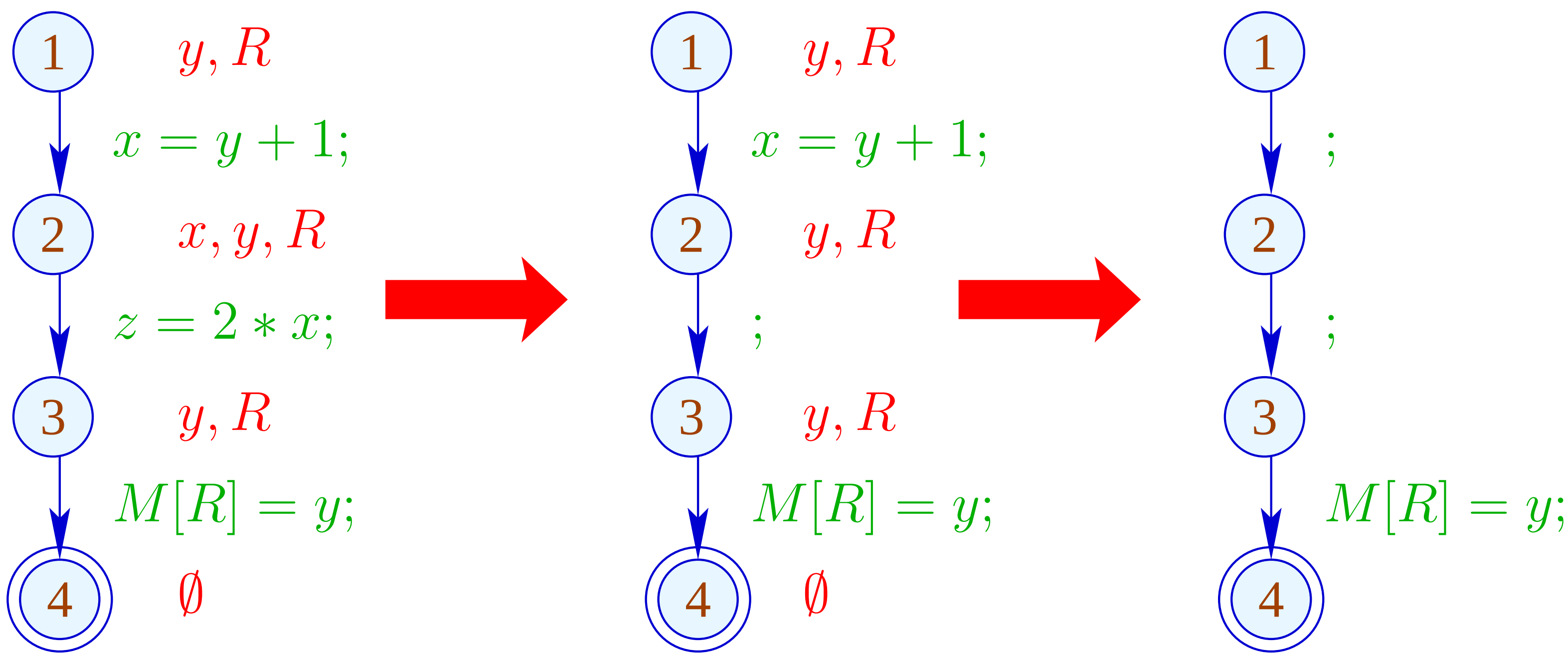
Removal of assignments to dead variables may kill further variables:



The left-hand side of no assignment is dead :-)

Caveat:

Removal of assignments to dead variables may kill further variables:



Re-analyzing the program is inconvenient :-(

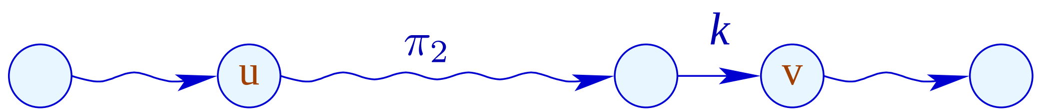
Idea: Analyze true liveness!

x is called truly live at u along a path π (relative to X), either

if $x \in X$, π does not contain a definition of x; or

if π can be decomposed into $\pi = \pi_1\, k\, \pi_2$ such that:

- k is a true use of x relative to π_2;
- π_1 does not contain any definition of x.

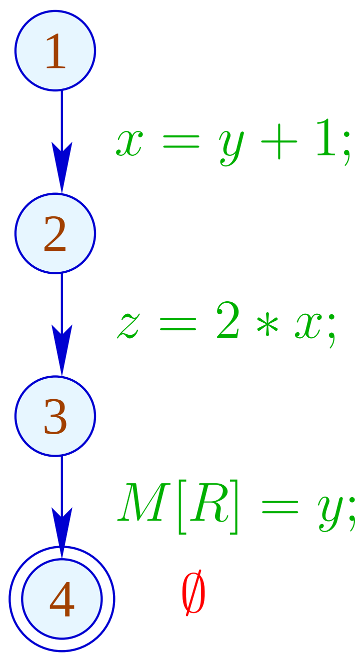


The set of truely used variables at an edge $k = (\_, lab, v)$ is defined as:

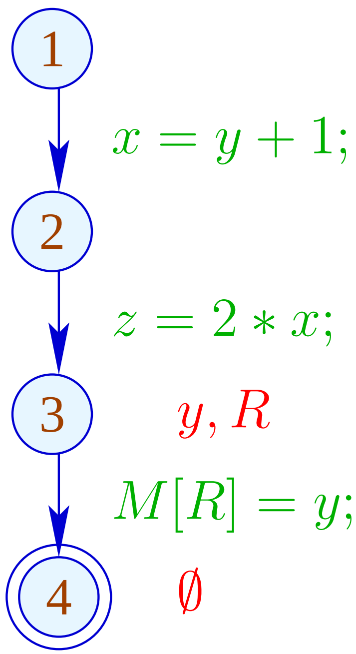
lab	truely used
;	$\emptyset$
Pos (e)	$Vars\ (e)$
Neg (e)	$Vars\ (e)$
$x = e;$	$Vars\ (e)$ $(*)$
$x = M[e];$	$Vars\ (e)$ $(*)$
$M[e_1] = e_2;$	$Vars(e_1) \cup Vars(e_2)$

$(*)$ – given that x is truely live at v w.r.t. π_2 :-)

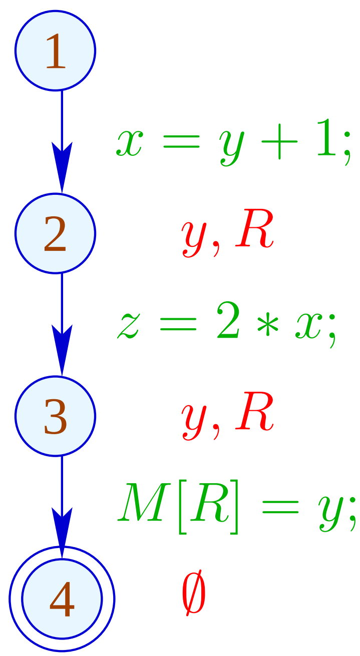
Example:



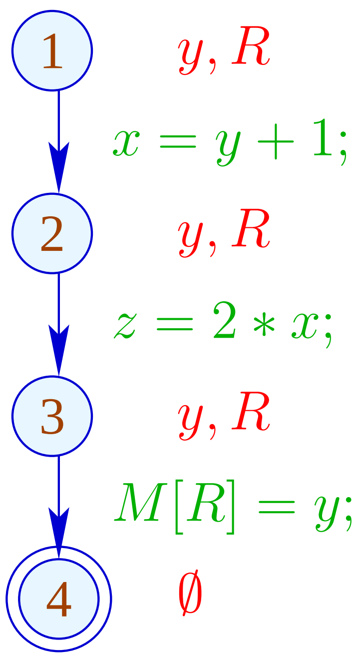
Example:



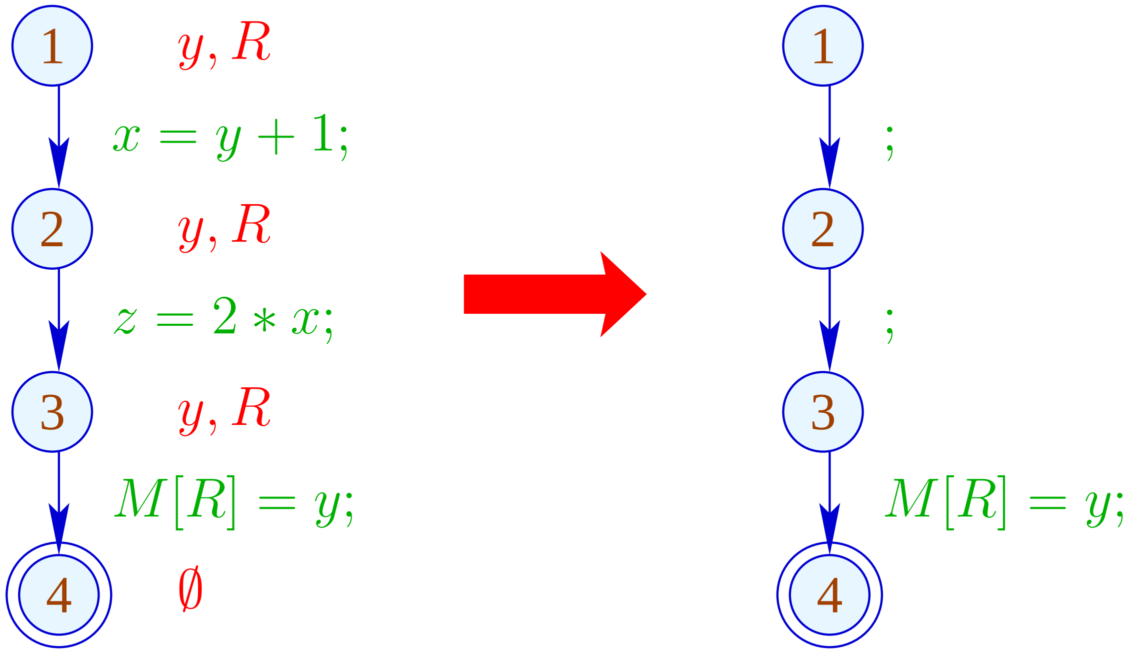
Example:



Example:



Example:



The Effects of Edges:

$$
\begin{array}{lcll}
[\![;]\!]^\sharp \, L & = & L & \\
[\![\text{Pos}(e)]\!]^\sharp \, L & = & [\![\text{Neg}(e)]\!]^\sharp \, L \;\; = & L \cup \mathit{Vars}(e) \\
[\![x = e;]\!]^\sharp \, L & = & (L \backslash \{x\}) \cup & \mathit{Vars}(e) \\
[\![x = M[e];]\!]^\sharp \, L & = & (L \backslash \{x\}) \cup & \mathit{Vars}(e) \\
[\![M[e_1] = e_2;]\!]^\sharp \, L & = & L \cup \mathit{Vars}(e_1) \cup \mathit{Vars}(e_2) &
\end{array}
$$

The Effects of Edges:

$$
\begin{array}{lcl}
[\![;]\!]^\sharp\, L & = & L \\
[\![\text{Pos}(e)]\!]^\sharp\, L & = & [\![\text{Neg}(e)]\!]^\sharp\, L \;\; = \;\; L \cup \mathit{Vars}(e) \\
[\![x = e;]\!]^\sharp\, L & = & (L \backslash \{x\}) \cup \; (x \in L) \, ? \, \mathit{Vars}(e) : \; \emptyset \\
[\![x = M[e];]\!]^\sharp\, L & = & (L \backslash \{x\}) \cup \; (x \in L) \, ? \, \mathit{Vars}(e) : \; \emptyset \\
[\![M[e_1] = e_2;]\!]^\sharp\, L & = & L \cup \mathit{Vars}(e_1) \cup \mathit{Vars}(e_2)
\end{array}
$$

Note:

- The effects of edges for truely live variables are more complicated than for live variables :-)
- Nonetheless, they are distributive !!

Note:

- The effects of edges for truely live variables are more complicated than for live variables :-)
- Nonetheless, they are distributive !!

  To see this, consider for $\mathbb{D} = 2^U$, $f\, y = (u \in y)\,?\, b :\ \emptyset$ We verify:

$$
\begin{array}{lll}
f\,(y_1 \cup y_2) & = & (u \in y_1 \cup y_2)\,?\, b :\ \emptyset \\
 & = & (u \in y_1 \vee u \in y_2)\,?\, b :\ \emptyset \\
 & = & (u \in y_1)\,?\, b :\ \emptyset \cup (u \in y_2)\,?\, b :\ \emptyset \\
 & = & f\, y_1 \cup f\, y_2
\end{array}
$$

Note:

- The effects of edges for truely live variables are more complicated than for live variables :-)
- Nonetheless, they are distributive !!

  To see this, consider for $\mathbb{D} = 2^U$, $f\, y = (u \in y)\, ?\, b : \emptyset$ We verify:

$$
\begin{array}{lll}
f\,(y_1 \cup y_2) & = & (u \in y_1 \cup y_2)\, ?\, b : \emptyset \\
 & = & (u \in y_1 \vee u \in y_2)\, ?\, b : \emptyset \\
 & = & (u \in y_1)\, ?\, b : \emptyset \cup (u \in y_2)\, ?\, b : \emptyset \\
 & = & f\, y_1 \cup f\, y_2
\end{array}
$$

$\Longrightarrow$ the constraint system yields the MOP :-))