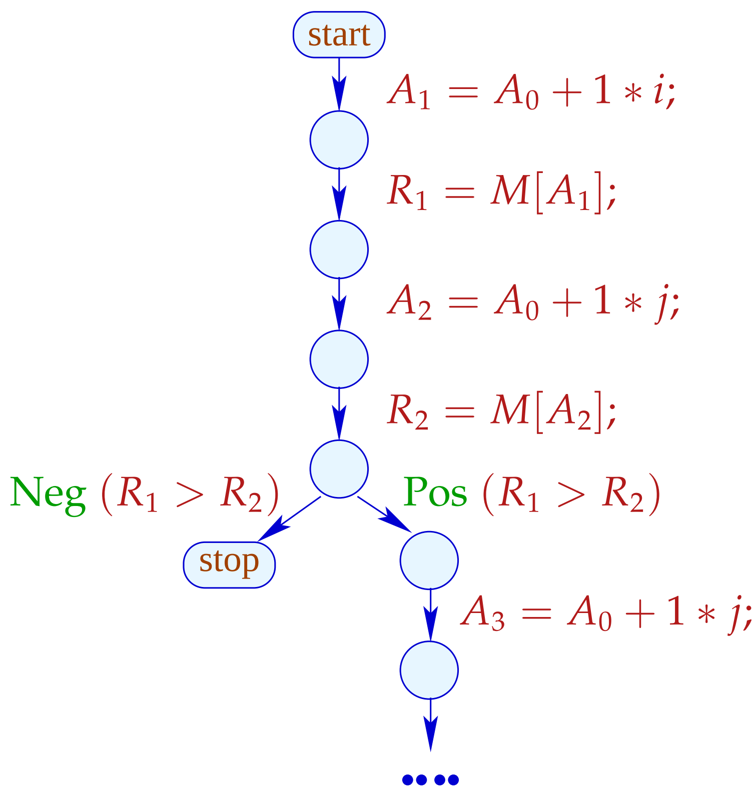
Background 1: An Operational Semantics

we choose a small-step operational approach.

Programs are represented as control-flow graphs.

In the example:



Thereby, represent:

vertex	program point
start	programm start
stop	program exit
edge	step of computation

Thereby, represent:

vertex	program point
start	programm start
stop	program exit
edge	step of computation

Edge Labelings:

Test :	Pos (e) or Neg (e)
Assignment :	$R = e;$
Load :	$R = M[e];$
Store :	$M[e_1] = e_2;$
Nop :	$;$

Computations follow paths.

Computations transform the current state

$$s = (\rho, \mu)$$

where:

$\rho : \mathit{Vars} \rightarrow \mathbf{int}$	contents of registers
$\mu : \mathbb{N} \rightarrow \mathbf{int}$	contents of storage

Every edge $k = (u, \mathit{lab}, v)$ defines a partial transformation

$$[\![k]\!] = [\![\mathit{lab}]\!]$$

of the state:

$$
\begin{array}{lcll}
[\![;]\!]\,(\rho,\mu) & = & (\rho,\mu) & \\
\\
[\![\mathrm{Pos}\,(e)]\!]\,(\rho,\mu) & = & (\rho,\mu) & \text{if } [\![e]\!]\,\rho \neq 0 \\
[\![\mathrm{Neg}\,(e)]\!]\,(\rho,\mu) & = & (\rho,\mu) & \text{if } [\![e]\!]\,\rho = 0
\end{array}
$$

$$
\begin{array}{lcll}
[\![;]\!]\,(\rho,\mu) & = & (\rho,\mu) & \\
 & & & \\
[\![\mathrm{Pos}\,(e)]\!]\,(\rho,\mu) & = & (\rho,\mu) & \text{if } [\![e]\!]\,\rho \neq 0 \\
[\![\mathrm{Neg}\,(e)]\!]\,(\rho,\mu) & = & (\rho,\mu) & \text{if } [\![e]\!]\,\rho = 0
\end{array}
$$

// $[\![e]\!]$: evaluation of the expression e, e.g.

// $[\![x+y]\!]\,\{x \mapsto 7, y \mapsto -1\} = 6$

// $[\![!(x == 4)]\!]\,\{x \mapsto 5\} = 1$

$$[\![;]\!]\,(\rho,\mu) \quad = \quad (\rho,\mu)$$

$$\begin{array}{lcll} [\![\mathrm{Pos}\,(e)]\!]\,(\rho,\mu) & = & (\rho,\mu) & \text{if } [\![e]\!]\,\rho \neq 0 \\ [\![\mathrm{Neg}\,(e)]\!]\,(\rho,\mu) & = & (\rho,\mu) & \text{if } [\![e]\!]\,\rho = 0 \end{array}$$

// $[\![e]\!]$: evaluation of the expression e, e.g.

// $[\![x+y]\!]\,\{x \mapsto 7, y \mapsto -1\} = 6$

// $[\![!(x == 4)]\!]\,\{x \mapsto 5\} = 1$

$$[\![R = e;]\!]\,(\rho,\mu) \quad = \quad (\boxed{\rho \oplus \{R \mapsto [\![e]\!]\,\rho\}}, \mu)$$

// where "$\oplus$" modifies a mapping at a given argument

$$
\begin{array}{lcl}
[\![R = M[e];]\!]\,(\rho,\mu) & = & (\boxed{\rho \oplus \{R \mapsto \mu([\![e]\!]\,\rho))\}}\,,\mu) \\[1em]
[\![M[e_1] = e_2;]\!]\,(\rho,\mu) & = & (\rho,\,\boxed{\mu \oplus \{[\![e_1]\!]\,\rho \mapsto [\![e_2]\!]\,\rho\}}\,)
\end{array}
$$

Example:

$[\![x = x+1;]\!]\,(\{x \mapsto 5\},\mu) = (\rho,\mu)$ where:

$$
\begin{array}{lcl}
\rho & = & \{x \mapsto 5\} \oplus \{x \mapsto [\![x+1]\!]\,\{x \mapsto 5\}\} \\
& = & \{x \mapsto 5\} \oplus \{x \mapsto 6\} \\
& = & \{x \mapsto 6\}
\end{array}
$$

A path $\pi = k_1 k_2 \ldots k_m$ is a computation for the state s if:

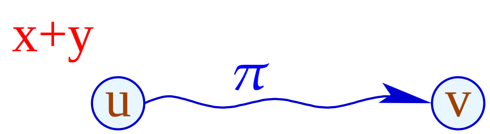
$$s \in \mathit{def}\,([\![k_m]\!] \circ \ldots \circ [\![k_1]\!])$$

The result of the computation is:

$$[\![\pi]\!]\, s = ([\![k_m]\!] \circ \ldots \circ [\![k_1]\!])\, s$$

Application:

Assume that we have computed the value of $x + y$ at program point u:



We perform a computation along path π and reach v where we evaluate again $x + y$ ...

Idea:

If x and y have not been modified in π, then evaluation of $x + y$ at v must return the same value as evaluation at u :-)

We can check this property at every edge in π :-}

Idea:

If x and y have not been modified in π, then evaluation of $x + y$ at v must return the same value as evaluation at u :-)

We can check this property at every edge in π :-}

More generally:

Assume that the values of the expressions $A = \{e_1, \ldots, e_r\}$ are available at u.

Idea:

If x and y have not been modified in π, then evaluation of $x + y$ at v must return the same value as evaluation at u :-)

We can check this property at every edge in π :-}

More generally:

Assume that the values of the expressions $A = \{e_1, \ldots, e_r\}$ are available at u.

Every edge k transforms this set into a set $[\![k]\!]^\sharp\, A$ of expressions whose values are available after execution of k ...

... which transformations can be composed to the effect of a path $\pi = k_1 \dots k_r$:

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

... which transformations can be composed to the effect of a path $\pi = k_1 \ldots k_r$:

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

The effect $[\![k]\!]^\sharp$ of an edge $k = (u, lab, v)$ only depends on the label lab, i.e., $[\![k]\!]^\sharp = [\![lab]\!]^\sharp$

... which transformations can be composed to the effect of a path $\pi = k_1 \ldots k_r$:

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

The effect $[\![k]\!]^\sharp$ of an edge $k = (u, lab, v)$ only depends on the label *lab*, i.e., $[\![k]\!]^\sharp = [\![lab]\!]^\sharp$ where:

$$\begin{array}{lclcl}
[\![;]\!]^\sharp\, A & = & A & & \\
[\![Pos(e)]\!]^\sharp\, A & = & [\![Neg(e)]\!]^\sharp\, A & = & A \cup \{e\} \\
[\![x = e;]\!]^\sharp\, A & = & (A \cup \{e\}) \backslash Expr_x & & \text{where} \\
 & & Expr_x \text{ all expressions which contain } x & &
\end{array}$$

$$
\begin{aligned}
[\![x = M[e];]\!]^\sharp\, A &= (A \cup \{e\}) \backslash Expr_x \\
[\![M[e_1] = e_2;]\!]^\sharp\, A &= A \cup \{e_1, e_2\}
\end{aligned}
$$

$$
\begin{aligned}
[\![x = M[e];]\!]^\sharp\, A &= (A \cup \{e\})\backslash Expr_x \\
[\![M[e_1] = e_2;]\!]^\sharp\, A &= A \cup \{e_1, e_2\}
\end{aligned}
$$

By that, every path can be analyzed :-)

A given program may admit several paths :-(

For any given input, another path may be chosen :-((

$$\begin{array}{lll} [\![x = M[e];]\!]^\sharp\, A & = & (A \cup \{e\}) \backslash Expr_x \\ [\![M[e_1] = e_2;]\!]^\sharp\, A & = & A \cup \{e_1, e_2\} \end{array}$$

By that, every path can be analyzed :-)

A given program may admit several paths :-(

For any given input, another path may be chosen :-((

$\Longrightarrow$ We require the set:

$$\mathcal{A}[v] \quad = \quad \bigcap \{ [\![\pi]\!]^\sharp \emptyset \mid \pi : start \rightarrow^* v \}$$

Concretely:

- $\longrightarrow$ We consider all paths π which reach v.
- $\longrightarrow$ For every path π, we determine the set of expressions which are available along π.
- $\longrightarrow$ Initially at program start, nothing is available :-)
- $\longrightarrow$ We compute the intersection $\Longrightarrow$ safe information

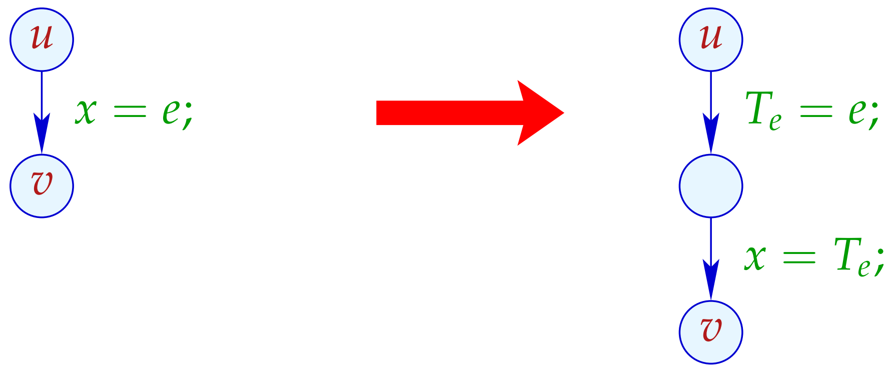
Concretely:

- $\longrightarrow$ We consider all paths π which reach v.
- $\longrightarrow$ For every path π, we determine the set of expressions which are available along π.
- $\longrightarrow$ Initially at program start, nothing is available :-)
- $\longrightarrow$ We compute the intersection $\Longrightarrow$ safe information

How do we exploit this information ???

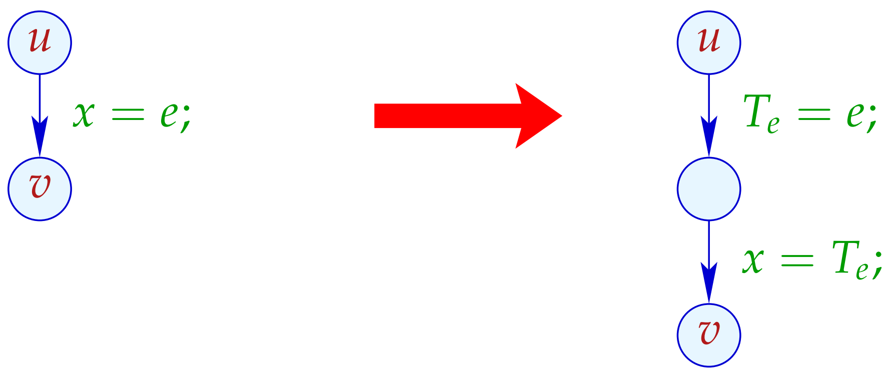
Transformation 1.1:

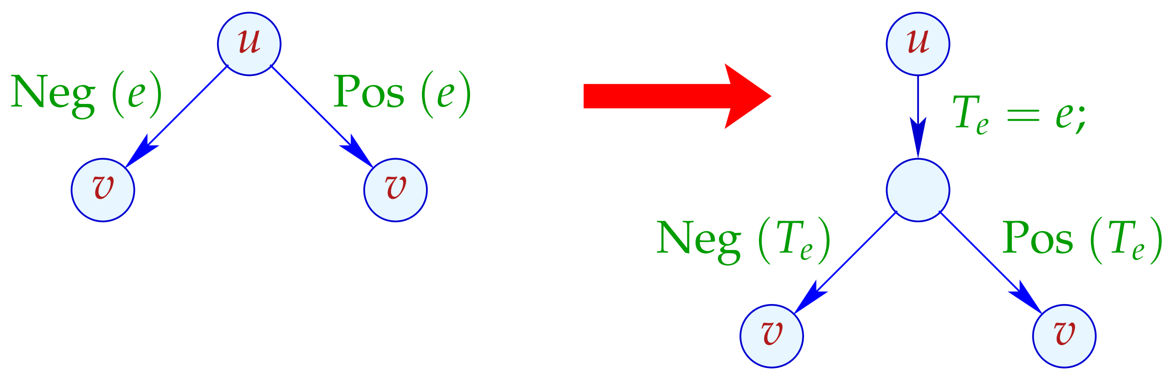
We provide novel registers T_e as storage for the e:



Transformation 1.1:

We provide novel registers T_e as storage for the e:

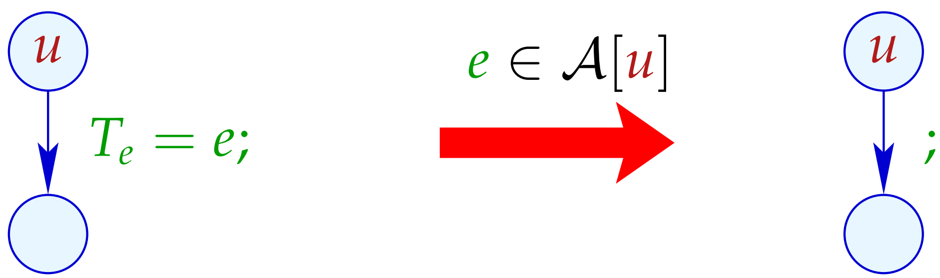




... analogously for $R = M[e];$ and $M[e_1] = e_2;$.

Transformation 1.2:

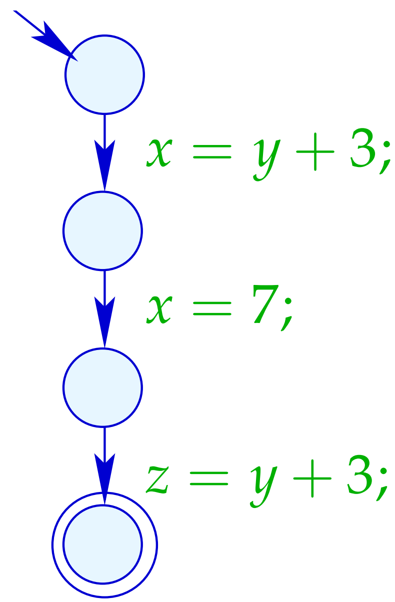
If e is available at program point u, then e need not be re-evaluated:



We replace the assignment with Nop :-)

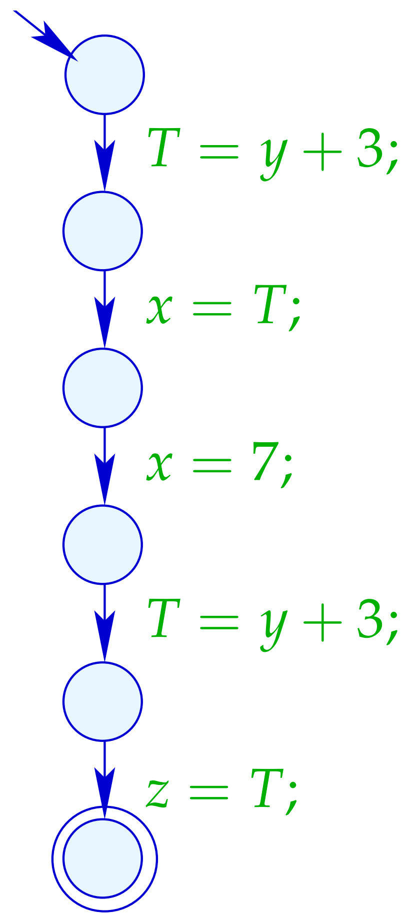
Example:

$$
\begin{aligned}
x &= y + 3;\\
x &= 7;\\
z &= y + 3;
\end{aligned}
$$



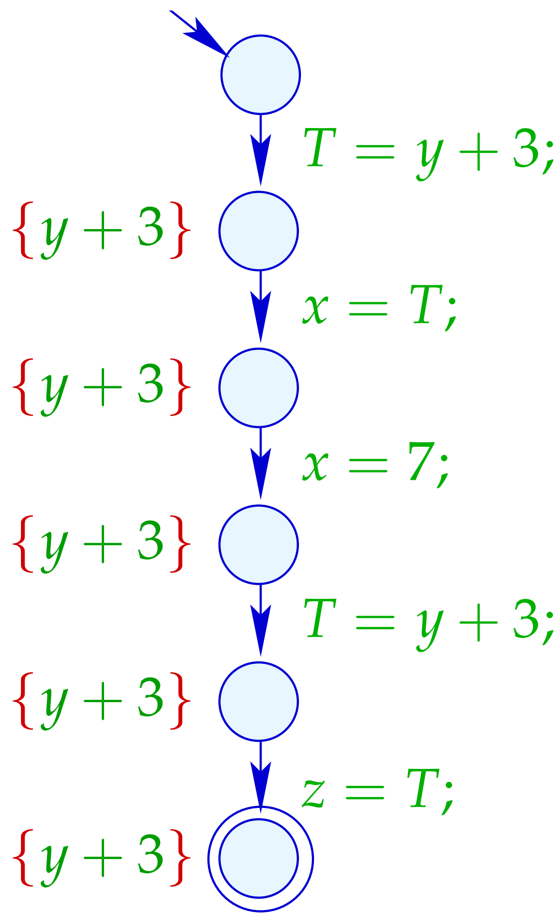
Example:

$$
\begin{aligned}
x &= y+3; \\
x &= 7; \\
z &= y+3;
\end{aligned}
$$



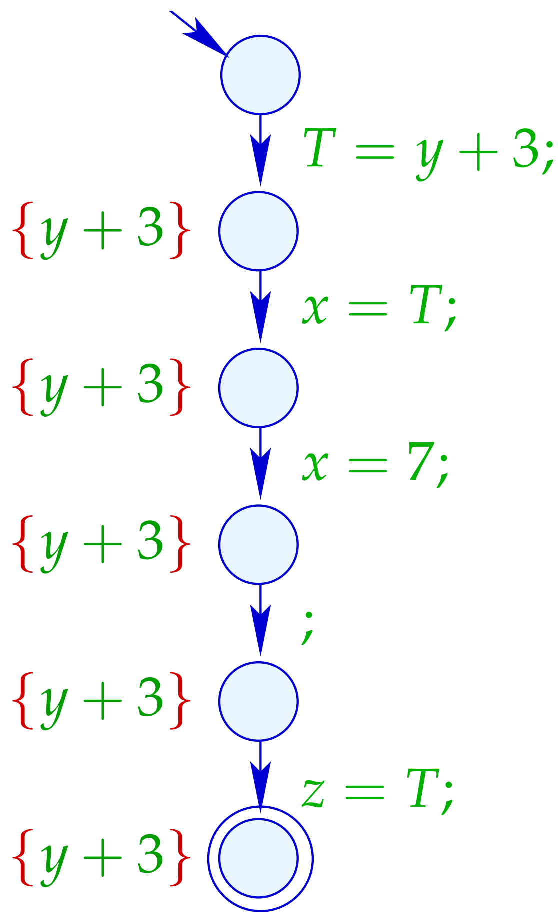
Example:

$$
\begin{aligned}
x &= y+3;\\
x &= 7;\\
z &= y+3;
\end{aligned}
$$



Example:

$$\begin{aligned} x &= y + 3; \\ x &= 7; \\ z &= y + 3; \end{aligned}$$



Correctness: (Idea)

Transformation 1.1 preserves the semantics and $\mathcal{A}[u]$ for all program points u :-)

Assume $\pi : \mathit{start} \rightarrow^* u$ is the path taken by a computation.

If $e \in \mathcal{A}[u]$, then also $e \in [\![\pi]\!]^\sharp\, \emptyset$.

Therefore, π can be decomposed into:

$$\text{start} \overset{\pi_1}{\rightsquigarrow} u_1 \xrightarrow{k} u_2 \overset{\pi_2}{\rightsquigarrow} u$$

with the following properties:

- The expression e is evaluated at the edge k;
- The expression e is not removed from the set of available expressions at any edge in π_2, i.e., no variable of e receives a new value :-)

- The expression e is evaluated at the edge k;
- The expression e is not removed from the set of available expressions at any edge in π_2, i.e., no variable of e receives a new value :-)

$$\Longrightarrow$$

The register T_e contains the value of e whenever u is reached :-))

Warning:

Transformation 1.1 is only meaningful for assignments $x = e$; where:

- $\longrightarrow$ $x \notin Vars(e)$;
- $\longrightarrow$ $e \notin Vars$;
- $\longrightarrow$ the evaluation of e is non-trivial :-}

Warning:

Transformation 1.1 is only meaningful for assignments $x = e;$ where:

- $\longrightarrow$ $x \notin Vars(e);$
- $\longrightarrow$ $e \notin Vars;$
- $\longrightarrow$ the evaluation of e is non-trivial :-}

Which leaves us with the following question ...

Question:

How do we compute $\mathcal{A}[u]$ for every program point u ??